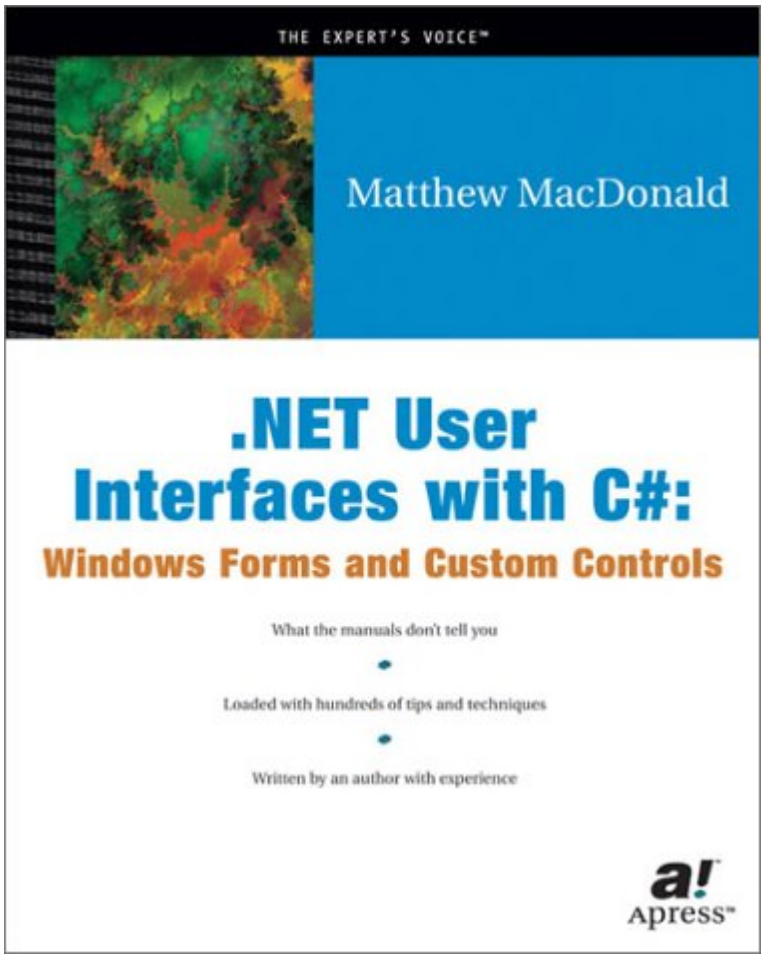
User Interfaces in C#-Windows Forms and Custom Controls

Matthew MacDonald


THE EXPERT'S VOICE™
Matthew MacDonald
.NET User Interfaces with C#:
Windows Forms and Custom Controls
What the manuals don't tell you
Loaded with hundreds of tips and techniques
Written by an author with experience
a!
Apress™

Apress™

Copyright © 2002 Matthew MacDonald

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mecha
prior written permission of the copyright owner and the publisher.

(pbk):
1-59059-045-7

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked
intention of infringement of the trademark.

Technical Reviewer: Gordon Henshaw
Editorial Directors: Dan Appleman, Gary Cornell, Jason Gilmore, Simon Hayes, Karen Watterson, John Zukowski
Managing Editor: Grace Wong
Copy Editor: Anne Friedman
Production Editor: Kari Brooks
Project Manager: Sofia Marchant
Compositor: Diana Van Winkle, Van Winkle Design
Artist: Kurt Krames
Indexer: Nancy A. Guenther
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski
Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 1001
Heidelberg, Germany.

User Interfaces in C#: Windows Forms and Custom Controls

byMatthew MacDonald ISBN:1590590457

Apress 2002 (586 pages)

Including a comprehensive examination of the user interface controls and classes in .NET, this resource provides an overview of how to design elegant user interfaces the average user can understand.

Companion Web Site

Table of Contents Back Cover

Table of Contents

User Interfaces in C#-Windows Forms and Custom Controls
Preface
Introduction
Chapter 1 - Creating Usable Interfaces
Chapter 2 - Designing with Classes and Tiers
Chapter 3 - Control Class Basics
Chapter 4 - Classic Controls
Chapter 5 - Forms
Chapter 6 - Modern Controls
Chapter 7 - Custom Controls
Chapter 8 - Design-Time Support for Custom Controls
Chapter 9 - Data Controls
Chapter 10 - MDI Interfaces and Workspaces
Chapter 11 - Dynamic User Interface
Chapter 12 - GDI+ Basics
Chapter 13 - GDI+ Controls

Preface

byMatthew MacDonald

Apress 2002

Companion Web Site

Preface

In the past few months, a deluge of .NET books has hit store shelves, each one eager to explain the new programming philo need to really master .NET programming. Part of the problem is that no single work can cover the entire .NET platform-a spra technology. Many .NET books provide a good overview of essential concepts, but they can't deal with the subtleties needed f

This book represents the start of the second wave of .NET books: closely focused works that give you the insight of experien *Custom Controls* takes a close look at all the ingredients you can use to design state-of-the-art application interfaces. It also framework for Windows. You don't just learn about anchoring and docking, you work with examples that show document-vie controls. You also learn how to design irregularly shaped forms, unshackle data binding, and build an integrated help systen

In short, this is the sort of .NET book that I would want to read as a professional developer. It's a book that goes beyond the generation of software applications. And seeing as you *are* reading this introduction, you've probably already realized that thi

Acknowledgments

This book has been, comparatively speaking, a lot of fun to write. As a result, I have a long list of people to thank.

Gary Cornell never ceases to amaze me with his ability to respond to emails mere seconds after they've departed from my c thanks to a number of other individuals at Apress who helped everything move swiftly and smoothly. They include Sofia Marc directly.

Gordon Henshaw performed the technical review for the C# edition of this book, and Gordon Wilmot performed the technical r and her unerring light touch helped guarantee the final polished product. I owe a heartfelt thanks to all.

Finally, I'd never write *any* book without the support of my loving wife, her parents, and my parents (who started this whole m

Introduction

byMatthew MacDonald

Apress 2002

Companion Web Site

Introduction

The .NET revolution is in full swing, and the confusion that has surrounded it for the last year is finally lifting. Developers are r development, cross-language interoperability, painless deployment, or a new way to access data. Instead, we now realize th miniature revolutions into one marketing term, along with a class library stocked with hundreds of pieces of prebuilt functiona

Unfortunately, you can't come to terms with the amazing breadth of the .NET Framework by reading a single book. To becom interest you the most. In other words, it's time to forget about the broadly sweeping goals of .NET for a moment, and focus o

This book explains how to program user interfaces applying the tools and techniques of the .NET world. You may already be interfaces, and the standard Windows controls and conventions). Other features are entirely new and will be unlike anything y all works through the .NET class library, which provides a new set of capabilities, subtleties, and quirks that every .NET prog

About This Book

User interface design deals with several aspects of programming. The tendency in a book about a topic like this is to pursue reference that only explains controls and commands is dangerous, and without a proper discussion of best practices and de enhance, debug, or scale up. For that reason, I've made the decision in this book to focus on three distinct themes.

What This Book Teaches You

This book fills three roles. It provides the following:

An overview of how to design elegant user interfaces the average user can understand. This is addressed

A comprehensive examination of the user interface controls and classes in .NET. Although this book is not a including providers, components, and custom controls.

A tutorial with best practices and design tips for coding user interfaces and integrating help. As a develope an entire user interface framework that's scalable, flexible, and reusable.

What This Book Doesn't Teach You

Of course, it's just as important to point out what this book *doesn't* contain:

A description of core .NET concepts like namespaces, assemblies, exception handling and types. These fundament including a number of excellent C# and VB .NET books from Apress.

A primer on object-oriented design. No .NET programmer can progress very far without a solid understanding of cla objects to encapsulate, organize, and transfer information.

A reference for Visual Studio .NET. The new integrated design environment provides powerful customization, automa control designers, for the most part it assumes that you already know how to use IDE to create controls and set pro

A comparison between .NET and its predecessors (including Visual Basic, C++, and even Java). Examining the evo master modern .NET development any faster. The best approach is to leave your past language allegiances behind.

If you haven't learned the .NET fundamentals, you will probably still be able to work through this book. You will probably nee issues along the way. On the other hand, if you have already read another, more general .NET book, you will benefit the mos

Note This book is targeted at experienced developers. If you have never programmed with a language like Visual Basic, C++/C#, or Java before, this isn't the place to start. Instead, start with an introductory book on object-oriented design or programming fundamentals.

Code Samples

It's a good idea to check the online site to download the most recent, up-to-date code samples. You'll need to do this to tes often left out. Instead, I focus on the most important conceptual sections so that you don't need to wade through needless e

Introduction

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter Overview

The following overview describes what each chapter covers. If you have some .NET experience, feel free to skip from chapter development it's probably easiest to read through the book sequentially, to make sure you learn the basics before encounter

Creating Usable Interfaces

User interface design is about more than just knowing how to program the latest trendy interface element-it's also about con the basics of interface design theory, and the principles that support every good design.

Designing with Classes and Tiers

For several years, programming books and articles have advocated a three-layered approach to application design that rigor follow these best practices, and programmers usually discover and rediscover that they are far more time-consuming and aw becomes dramatically easier with .NET-and how it might work for you.

Control Class Basics

This chapter delves into the details of one of .NET's most feature-rich classes: the Control. In this chapter, you learn how the control relations, and handling Windows messages. You also learn about some of the basic System.Drawing ingredients for

Classic Controls

The classic controls include basic tools for input, selection, and display that have been used since the ancient days of 16-bit controls, and the hyperlink label. It rounds up with demonstrations of control validation and drag-and-drop techniques.

Forms

The Form class is the basis for every application window in a .NET program. To use forms effectively, you need to understar considers exciting new techniques like visual inheritance, Windows XP styles, and irregularly shaped forms. It also explains

Modern Controls

This chapter dissects everyone's favorite Windows controls, including TreeView, ListView, ToolBar and StatusBar. As these useful higher-level features or are tailored for a specific type of data.

Custom Controls

Custom control development is one of the key themes of this book, and a remarkable feature of the .NET platform. This cha thumbnail viewer, a progress user control, and a directory tree. It also considers advanced topics like asynchronous control p latched onto any .NET control.

Design-Time Support for Custom Controls

Creating a custom control is easy, but making it behave well in the design-time environment often takes a little extra wizardr equip your controls for Visual Studio .NET. You also tackle different models of custom control licensing.

Data Controls

Most applications need to deal with data at some point. This chapter considers how you can integrate data into your user in

Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 1: Creating Usable Interfaces

Overview

Sometimes it seems that no one can agree what user interface design really is. Is it the painstaking process an artist goes t usability lab subjecting users to a complicated new application? Is it the series of decisions that determine how to model info

In fact, user interface design is really a collection of several different tasks:

User interface modeling. This is the process where you look at the tasks a program needs to accomplish, and de to combine instinct, convention, a dash of psychology, and painstaking usability testing.

User interface architecture. This is the logical design you use to divide the functionality in your application into se portions of the user interface framework.

User interface coding. This is the process where you write the code for managing the user interface with the appr model and architecture before you begin this stage.

This book concentrates on the third, and most time-consuming step, where user interfaces designs are translated into code design, and discussion about user interface architecture, the second item on the list, recurs throughout this book (and is the

This chapter, however, focuses on the first task: user interface design. Here you'll examine the essential guidelines that no p entering into the mind of that often-feared final judge: the end user.

You could skip ahead at this point and dive right into .NET code. However, the greatest programming framework in the worlc less important than learning how to work with it in code.

Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Why Worry About the Interface?

User interface is the thin outer shell that wraps a program's logic and provides a way for ordinary users to interact with it. Us

- Interpreting what a user wants and translating it into the corresponding operations.
- Retrieving information and displaying it in different ways.
- Guiding users through a task (and steering them away from common mistakes).

User interfaces bear the weight of a program, because they are the only part the user interacts with. It doesn't matter what y pianist in a small, dark prison cell. As with anything else, people judge and identify programs based on what they can see fr look. Ugly and confusing interfaces, on the other hand, lead to a legacy of headaches for developers and end users.

In programming circles, user interfaces are often the subject of heated debate. Some developers resent the whole topic of us requirements, the hard-to-please end users, and the perception that they have to simplify the product of their perfectly natura fad. They aim to discover the newest and most avant-garde user interface controls before they have been adopted as standa

Ultimately, both approaches are bad news for end users, who just want a simple, unobtrusive interface that works exactly th the unwritten rules of user interface design.

Tip

Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

A Brief History of User Interfaces

You might think that user interface design is a history of continuous innovation. In fact, user interface design is actually mark some unpredictable time, a lone programmer or innovative programming team creates a truly new user interface model that incompatible versions. This process of false imitation continues until the next revolution.

So what are these eras of user interface development? It all began very simply.

The Command-Line Era

Almost everyone who has any experience with computers has at least glimpsed the fabled command line. Today's novice us from most people. Even advanced computer users are sometimes bound by the superstition that a command line lurks behi

A command-line interface is the power user's dream. Of course, even power users have to learn somewhere, and most forge

The command line is, in many respects, the original way of doing things, and it's arguable that it's not so much an interface began as the basis for operating systems like DOS (see Figure 1-1) and UNIX, were the basis for early database application

Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\Matthew>cd..
C:\Documents and Settings>dir
Volume in drive C is Applications
Volume Serial Number is 38F8-94BF
Directory of C:\Documents and Settings
2001/10/26 05:04 PM <DIR> .
2001/10/26 05:04 PM <DIR> ..
2001/02/11 11:47 AM <DIR> Administrator
2001/11/15 07:27 PM <DIR> All Users
2002/01/28 10:17 PM <DIR> Faria
2002/01/31 08:44 AM <DIR> Matthew
0 File(s) 0 bytes
6 Dir(s) 1,803,735,040 bytes free
C:\Documents and Settings>_

Figure 1-1: The archetypal command-line interface

For example, the Visual Studio .NET interface provides a Command Window that lets you interact with the IDE or execute s (like auto-complete), it's still a basic command-line interface (see Figure 1-2).

Figure 1-2: The command line in Visual Studio .NET

Command-line interfaces are characterized by the following traits:

- **Ultimate control.** Users can do anything in any order, so long as they remember the "secret codes."
- **Ultimate lack of structure.** Users not only have to remember what to do, but what order to do it in. In DOS, just mo assumes that each operation is atomic, and it doesn't associate one task with another.
- **A "hands off" treatment of user.** With a few minor exceptions, there's no helpful prompting, tips, or guidance.
- **No metaphors.** This makes it easy to grasp the basic process (type in words, press Enter), which never changes. F example, if you know how to copy a file in UNIX, you still don't know how to delete it.)

Today, a command-line model could still turn up in one of your user interfaces, but it's unlikely

Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Creativity vs. Convention

Many user interface projects are sidetracked when they meet up with the developer's need for creativity. Unfortunately, an a
from the user's point of view, how it works).

Ask yourself this question: would car manufacturers allow the same degree of creativity that some developers take in applica
impossible to imagine a car manufacturer taking the same kind of liberties. Every year, new car models appear that have be
modern. It doesn't take much inspection to see that the air conditioners and radios always work almost exactly the same as
configuration is generally unchanged; and the controls for starting, stopping, and slowing the car down are indistinguishable.

Even in the better applications of today, this consistency is rare. If you install a new program on your computer, are you con
printer or give you a chance to tweak some setting first? And exactly where do you find the menu command for that all-impor

Tip On the good side, some conventions are well followed (like using Esc to exit a dialog box). Other conventions have taken over just because Microsoft enforces them, like the editing commands built in to standard controls like the text box, and the way you resize or move a window.

To make a long story short, convention is the way that users learn to work with a variety of software. Violating convention be
the amount of information a user needs to know to use computer software.

Consistency in .NET

Microsoft has made no secret that one of its goals with the .NET platform is to make the programming model more consiste
set of data types and functionality drawn from a shared class library. You can see this in the lavish use of interfaces and inh
You can even see this in the way Visual Studio .NET allows you to use its powerful debugging tools, regardless of whether y

In short, if consistency is so prized by cutting-edge software developers, why would anyone assume it's not just as importan

The "Act-Like-Microsoft" Principle

Windows developers have it rather easy. The secret to making a program that the average user can understand, and even e
the best choices in their applications-but for the most part, that isn't important. If the users of your application have ever use
Explorer. In fact, if your users are regular computer users, they probably spend the majority of their computing time with Wo

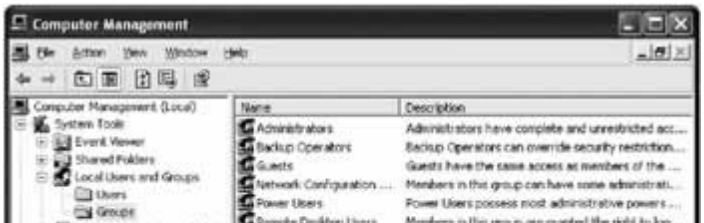
There's rarely a good reason for deviating from Microsoft standards. If average users have learned anything, it's the commor
pour ridiculous amounts of money into extensive usability tests, suggesting that their designs might not only be more recogn

If you aren't creating an office productivity or document-based application, you should still pay careful attention to Microsoft's
Internet, and reading email). In some cases, you might need to investigate another application (like Adobe Photoshop in the

Tip Remember, when you follow expected conventions, you don't just make it easier for users to learn your application.You also help train those users for the next programmer's software.

Administrative Utilities

One good example of a Windows convention is found in Microsoft's design of system and management utilities. These utiliti
Windows 2000 and later operating systems, Microsoft uses this design everywhere it can, even stretching the convention to



Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Handling Complexity

Some programmers (and many more management types) believe the myth that when users complain that an application is t
often just to slap a new button somewhere that will supposedly make it quicker to access features and thus render the progr

For example, consider the sample audio recorder and its "improved" version, both shown in Figure 1-6. It may be a little quic

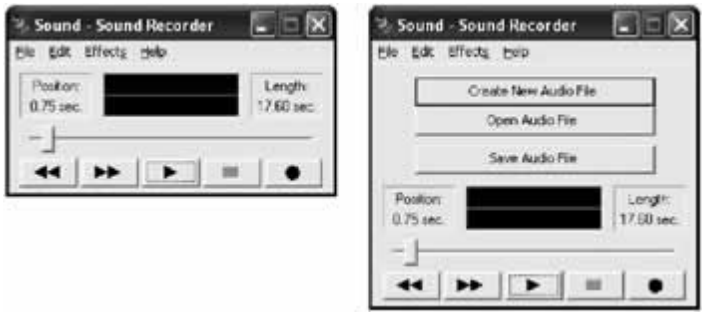


Figure 1-6: Two approaches to an audio recorder

In reality, when a user complains that an interface is confusing, it's rarely because it lacks a few quick shortcut controls or ti
Adding more buttons to the audio recorder doesn't just make the interface look ugly; it also makes it seem impossibly comp

Segmenting Information

Deciding how to divide a product's functionality into separate applications, windows, and controls is the most important user
similar management windows. For example, a database application might have an add/remove/configure window for configu
that steps through multiple steps leading to a single goal. Before beginning an application, you should identify the most obvi

Some other principles are outlined here:

- Use the common Windows analogies. These are "obvious" metaphors (for example, document icons represent files)
- Don't let metaphors take over your program. For example, you shouldn't find a cute way to reuse a metaphor when
  trash can to delete files *and* eject floppy disks.)
- Use the right controls to offload the work. Controls like the TreeView, ListView, and DataGrid can handle the basic u
- Hide unnecessary information.
- Appearing complex is being complex. A program appears logical when it does what the user expects. Keep this in r

Inductive User Interface

Microsoft has a new methodology designed to make user interfaces simpler by breaking features into individual self-explana
incorporates a set of tasks related to a single type of information. This type of interface, geared for the lowest (and most com
current example of inductive user interface (IUI) design is Microsoft Money 2000.

IUI is in its infancy. No clear conventions exist, and it's fairly labor intensive to design. For most programmers it makes sens
support). You can read the initial IUI guidelines in the MSDN (online at http://msdn.microsoft.com/library/en-us/dnwindev/htm

Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Helpful Restrictions

Most programmers fall into the category of "power users" of computer systems. It's for that reason that it sometimes comes impose restrictions. To a developer, restrictions often seem to run contrary to the goal of application programming- they mak overall abilities of your program, but you increase the efficiency and confidence of the average user.

Restricting the User's Ability to Make a Mistake

If you aren't careful, a great deal of code can be wasted attempting to detect and deal with errors. The problem is that once a matter how carefully worded or helpful the error message attempts to be, it's likely to make the user feel foolish, guilty, and the message appears to clear it from the screen, and then try the same thing over again.)

It doesn't matter whether you display this message after the user clicks the OK button or (worse yet) as soon as a field lose

A better approach is to spend your energy preventing errors from happening in the first place. For example:

Limit the number of characters a text box can accept, and use the key press event to make sure invalid characters a
Use drop-down lists when the user is selecting one of several predefined choices.
Disable (or "grey out") invalid options. In the case of a complex application with many menu options and toolbars, y elements. You see examples of both techniques in later chapters.

Tip Many of these options represent a tradeoff between usability and maintainability. For example, enforcing field length restrictions in a text box can cause quite a headache if the allowed length of the underlying database field changes. A better approach may be to dynamically determine the length of the field from the database when the form first loads. This ensures that you won't need to recompile your code when the database changes, but it also forces you to write (and test) additional code.

Restricting the User's Choices

Another common myth in user interface programming is that the more advanced an application is, the more options it should feature, you should do both, and allow the user to choose. Unfortunately, this type of logic (deciding not to decide) is shirking the application, and may not even know that a configuration option is available or how it works. Adding more options dramat installations.

The basic rule is that if something appears more complicated, it *is* more complicated. Adding gratuitous options can make s a microwave. If microwave users had to navigate through a series of menus that gave options about the pitch of the "food rea would suddenly become much more intimidating. Even more practical enhancements, like allowing the user to fine-tune pow added complexity.

Heavily customizable applications also bury genuinely useful options in a slew of miscellaneous, less important properties. F of an application as you add extraneous elements. Most options can either be eliminated and handled by a reasonable defau every time you give a user an option you are forcing the user to make a decision. Many users become increasingly unsettle

Restricting the User's Imagination

If you've ever worked at a Help desk, you probably understand that the human mind thinks in terms of cause and effect. The for mysterious problems or unexpected behavior with their applications, even if these explanations seem wildly fantastical to

When designing a program, you need to restrict this natural tendency. Some ways you can do this include:

Give feedback for long tasks. Some possibilities include a continuously updating dialog box message, progress bar
Show; don't tell. The average user generally views long-winded dialog boxes that explain what will happen next with just direct the user to an important area of the screen). For example, many drawing programs now use thumbnail pr
Avoid the super-intelligent interface. People love to see the demon in the machine. Even in a painstakingly designed confound users of all levels. Don't assume your application can determine what the user intends to do. Automatic fi

Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev Next

Programming User Interface for the Web

.NET provides web controls that resemble their common Windows counterparts, even maintaining their state automatically a sometimes be transferred from one environment to the other. With new features like disconnected data access, you can eve

There are still some restrictions inherent to the world of HTML. Most significant, HTML is not a windowing system. There's n Because of these limitations, it's extremely difficult to create some application types that are easy for desktop applications, developers will start to create custom .NET menu controls that can render themselves as client-side DHTML, but for now you

The part of the .NET framework that allows you to create web applications is ASP.NET. ASP.NET elegantly solves some lon react to an ASP.NET control event, you need to trigger a postback, which sends the page back to the server. This takes a s changes, and impossible to use capture events like mouse movements or key presses. For reasons like this, you can't perfo all the information is entered and the page is submitted. ASP.NET also introduces data binding as a key technique. It works considerations. Finally, you should also be aware that there is little standardization in the Internet world. Most users can agr is accused of copying, not praised for following convention.

Prev Next

Chapter 1 - Creating Usable Interfaces

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

User interface is really a blend of common sense, bitter experience, and a little luck. Many other books treat the subject in m
for Programmers, a short and insightful book from Apress.) There are also seminal works from Microsoft on Windows conve
and no longer reflects modern controls and Microsoft's latest trends. Parts of *Microsoft Windows User Experience* can be rea

A large part of this chapter has focused on a back-to-basics approach that stresses organization and logic instead of graphi
anywhere it if looked like Microsoft Excel. The dividing line is usually drawn between productivity applications and entertainm
market for skinnable word processors isn't nearly as large.

Now that you have a basic understanding of what makes an interface truly usable, it's time to shift your focus to the underly
tiers have to do with user interface programming, and how .NET and C# let you work with them.

Chapter 2 - Designing with Classes and Tiers

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 2: Designing with Classes and Tiers

Overview

Some developers resent user interface programming because they believe it's all about painting icons, rewording text, and e
Certainly, making a program usable, elegant, and even impressive (or cool) is no trivial task. It can even make the difference
tirelessly. This kind of excitement about a program's look and feel has driven obscure products like Kai's Power Tools and ev
in creating and maintaining sophisticated enterprise-level applications realize that there is another set of design considerati

Application architecture determines how a user interface "plugs in" to the rest of an application. Today's development softwa
developers usually spend little or no time thinking about it. User interface code is usually inserted wherever is most immedia
tightly bound to a particular problem, scenario, or data source, and heavily interwoven with the rest of the application logic. T
with anything more than trivial changes. To make the jump from this type of scattered user interface coding to a more mode
an entire interrelated framework.

This chapter explains how the principles of object-oriented programming and three-tier architecture apply to user interface de

- How controls and windows use objects.
- Why inheritance is more important for user interface design than for business logic.
- How the .NET framework approaches user interface design, and how it affects your coding practices.
- What makes an interface data-driven and well encapsulated.
- What the dream of three-tier design promises, why it's so hard to achieve, and whether other solutions are possible.

The emphasis in this chapter is on general concepts. You'll see some code, but you won't learn about the intricate details lik
controls and user interface coding in the chapters that follow.

Chapter 2 - Designing with Classes and Tiers

byMatthew MacDonald

Apress 2002

Companion Web Site

Classes and Objects

Today, it's generally accepted that the best way to design applications is by using discrete, reusable components called *ob*

A typical .NET program is little more than a large collection of class definitions. When you start the program, your code crea
are defined in other referenced assemblies and in the .NET class library (which is itself just a collection of useful assemblies

The Roles of Classes

It's important to remember that although all classes are created in the same way in your code, they can serve different logica

Classes can model real-world entities. For example, many introductory books teach object-oriented programmin
directly correspond to an actual *thing* in the real world.

Classes can serve as useful programming abstractions. For example, you might use a Rectangle class to store
WinMessage class to hold information about a Windows message. These classes don't need to correspond to tangi
your code. Arguably, this is the most common type of class.

Classes can collect related functions. Some classes are just a collection of static methods that you can use with
functions, and might have names like GraphicsManipulator or FileManagement. In some cases, a helper class is jus
related objects. In other cases it's a useful way to create a repository of simple routines that can be used in a variet

Understanding the different roles of classes is crucial to being able to master object-oriented development. When you create
giving it more than one type of role. The more vague a class is, the more it resembles a traditional block of code from a non-o

Classes and Types

The discussion so far has reviewed object-oriented development using two words: classes and objects. Classes are the defi
design is that you can put any class to work, and use it to create as many objects as you need.

In the .NET world, however, these terms are a little blurred. The .NET class library is really built out of *types*, and classes are
types and how they can be used. If you need to refresh your memory and get reacquainted with .NET's object family, use th

Structures

Structures are like classes, but are generally simpler. They tend to have only a few properties (and even fewer important me
This means that structures act differently in comparison and assignment operations. If you assign one structure variable to a
compare structures, you are comparing their contents, not the reference.

```
structureA = structureB; // structureA has a copy of the contents of structureB.
                         // There are two duplicate structures in memory.

if (structureA == structureB)
{
    // This is true as long as the structures have the same content.
    // Using full binary comparisons like this can slow down performance.
}
```

Some of the structures in the class library include Int32, DateTime, and graphics ingredients like Point, Size, and Rectangle

Chapter 2 - Designing with Classes and Tiers

byMatthew MacDonald

Apress 2002

Companion Web Site

Interacting with a Control

One interesting and little-known fact about .NET controls is that they provide two different forms of interaction. The first, and

For example, imagine you have a text box that's designed for numeric entry, and you want to examine every key press to m customized text box, and override the OnKeyPress() method to add this extra verification logic.

```
public class NumericTextBox : System.Windows.Forms.TextBox
{
    protected override void OnKeyPress(KeyPressEventArgs e)
    {
        base.OnKeyPress(e);
        if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar))
        {
            e.Handled = true;
        }
    }
}
```

The OnKeyPress() method is invoked automatically by the Windows Forms engine when a key is pressed in a TextBox cont isn't, the Handled flag is set to true, which cancels all further processing, effectively making sure that the character will neve

This design pattern is useful if you use a number of controls with extremely similar behavior. If, on the other hand, you need consider a button control. You could react to a button click by creating a special class for every button on your application, a encapsulated, it would quickly become completely disorganized, swamped by layers of button classes that have little to do v

The View-Mediator Pattern

The approach that .NET encourages circumvents this problem by using events. When you create a form in Visual Studio .NI handling logic is inserted as a series of special methods, which are added to the custom form class.

In other words, every form acts as a giant switchboard for all the controls it contains. This type of design pattern, which is so View-Mediator pattern. It dictates that one central class organizes each individual window.

Here's the same text box example you saw earlier, rewritten as form-level event handler. The event handler is hooked up in t InitializeComponent() method).

```
public class MainForm : System.Windows.Forms.Form
{
    System.Windows.Forms.TextBox txtUserName;

    public MainForm()
    {
        txtUserName = new System.Windows.Forms.TextBox();
```

Chapter 2 - Designing with Classes and Tiers

byMatthew MacDonald

Apress 2002

Companion Web Site

Encapsulation

Encapsulation is the principle that suggests objects should have separate, carefully outlined responsibilities. Everything they automatically, where possible. Encapsulation is often identified as a pillar of object-oriented programming, but it has played a performs a discrete well-identified task independently, and has a much better chance of being reused in another application (

The best way to start separating your user interface code is to think more consciously about encapsulation. The custom for presents a danger. It potentially encourages you to mix in a great amount of additional material through the form's event han

Of course, there are finer points to perfecting encapsulation. Custom controls, which handle some basic user interface opera you keep encapsulation in mind. These techniques are not all immediately obvious. In this book, I'll return to the principle of

Use Enumerations and Resource Classes

User interface controls often require sets of constants, and trying to hard-code them is a tempting trap. Instead, you should example, you can define enumerations that help you manage and identify different levels of nodes in a TreeView control (se your program. Extraneous details like SQL statements should also be strictly confined to distinct resource classes.

Use Collections

Objects are only as good as the way you can access them. On its own, a data object is a group of related information. By u an entire set of complex data, making it easier to share with other parts of your program.

Restrain from Sharing Control References

It's easy to pass control references to helper methods. For example, you can create utility classes that automatically fill com interface tasks, can make it extremely difficult to make even simple modifications to the user interface. As a rule of thumb, b

Define a Data Transfer Plan

The single greatest challenge when creating a reusable object framework is deciding how to retrieve data and insert it into th need to tolerate slightly long-winded code. You also need to create a clearly defined plan for transferring data that can be sh perfect, nearly universal solution for transferring information.

Use a Central Switchboard

The form acts as a switchboard for all the controls it contains. However, you shouldn't rely on forms for anything more. Instea much logic. To accomplish this, forms should always hand off their work to another central switchboard, like an application a new object in the form code and calling a single method. However, if you add another layer of indirection by forcing the for little more independence and gets closer to the ideal of three-tier design. Figure 2-5 shows how this process might work whe calls a DoCustomerUpdate() form method, which then calls the required methods in the CustomerDB business object (and

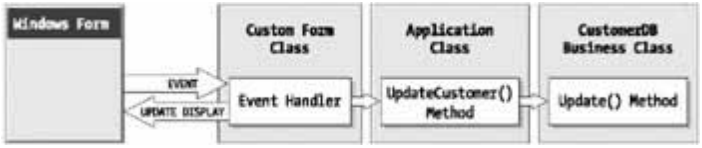


Figure 2-5: Using form and application switchboards

Create Data-Driven User Interfaces

Chapter 2 - Designing with Classes and Tiers

byMatthew MacDonald

Apress 2002

Companion Web Site

Developing in Tiers

The basic principle of three-tier design is simple. An application is divided into three distinct subsystems. Every class belong identified as:

A presentation layer, which converts a user's actions into tasks and outputs data using the appropriate controls.
A business layer, where all the calculations and processing specific to the individual business are carried.
An underlying data layer, which shuttles information back and forth from the database to the business objects.

An object in one tier can only interact with the adjacent tiers, as shown in Figure 2-6.

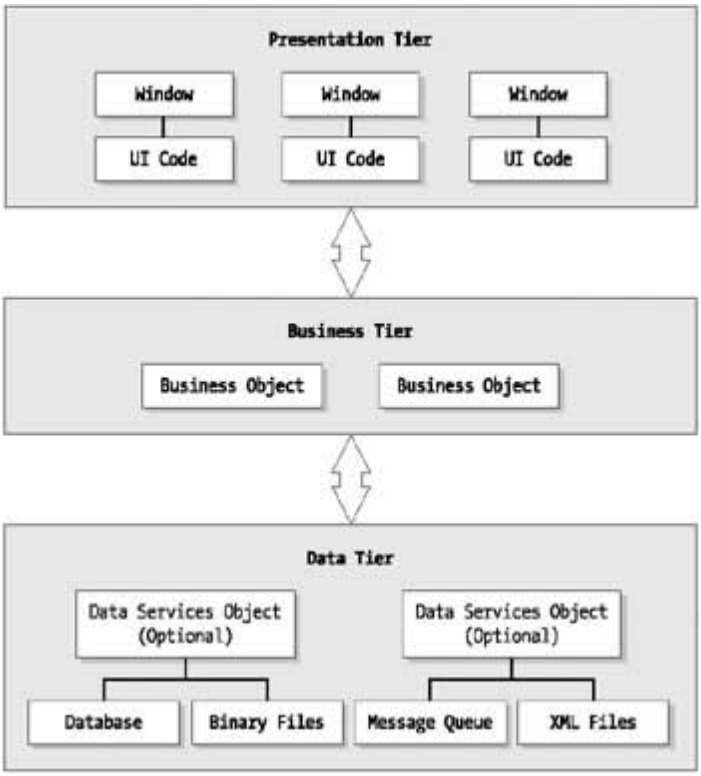


Figure 2-6: Three-tier design

Almost everyone agrees that this sort of structure is the best way to organize an application, but it's not always easy to impl complicated, and sometimes make assumptions or have expectations about the way they will receive information. The result

Problems with Three-Tier Design

Before you can solve the problems of three-tier design, you have to understand what these problems are. The problems, alth

The presentation tier

Though it doesn't explicitly state it, three-tier design really requires a fair degree of consistency among user interface contro change-like substituting a ListView control for a DataGrid-requires a totally different access model. DataGrids are filled exclu other ListView columns, you have to add a collection of fields to each individual item. These quirks are easy enough to mas that can quickly and efficiently fill common controls.

The business tier

Chapter 2 - Designing with Classes and Tiers

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

Three-tier design is an abstraction. No successful application will implement it exactly. However, it's also a powerful guidelir about DataSets and business objects in a book on user interface design. (In fact, there are other excellent .NET books writt interface tier can communicate with other parts of your program, you start to make the transition from a simple collection of

The following chapters of this book focus on mastering the tools of .NET user interfaces. Considerations for design and user model of .NET. This chapter was designed to give you the background in application architecture that you need to understar discover an entirely new and richer way to program user interfaces.

This chapter also introduced you to the broad picture of user interface in the .NET world, and the basic design assumptions itself allows you a considerable amount of freedom to create the exact framework that you want. In later chapters you'll learn

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 3: Control Class Basics

This chapter explores the Control class, which provides basic functionality for the family of Windows controls. You will learn I'll also introduce some important pieces of the System.Drawing namespace that allow you to create structures that represe

The Windows Forms Package

.NET provides two toolkits for user interface design: one for web applications, and one for Windows development. This chap interfaces found in everything from office productivity software to arcade games. The one detail that all these applications ha present information and receive user input.

It's easy to imagine that "Windows Forms" refers to a special part of the .NET class library, where fundamental classes like is the technology that allows the Common Language Runtime to interact with control objects and translate them into the low and windows, and the Common Language Runtime handles the details like routing messages, keeping track of window hand

This idea isn't new. In the past, developers have used the MFC framework in C++, WFC in J++, and Visual Basic's own "Rub These frameworks all provide an object-oriented wrapper around the Windows API (which, on its own, is a disorganized colle have all suffered from a few problems.

Lack of consistency. If you learn how to use MFC, you still won't know anything about creating Visual Basic user i dramatically different object models and philosophies.

Thin layer/thick layer problems. Frameworks tend to be either easy to use, or powerful, but not both. MFC is real Visual Basic developers have the benefit of a simple framework, but face the lingering dread that they will need to d

Subjugation to Windows API rules. The Windows API dictates certain harsh realities. For example, once you cre on how the Windows API is organized, but they often lead to confusing inconsistencies in a framework's object mod

The result of these limitations is that there are essentially two types of frameworks: those that are complicated to use for si for complex tasks (like VB). These object models provide a modern way to code user interfaces, but many programmers wo

The .NET Solution

.NET addresses these problems by becoming more ambitious. The result is a user interface framework that uses some inno some examples- tasks that .NET can perform but the Windows API cannot:

Change fixed style properties like the selection type of a list box or the border type of a window.
Change a form's owner.
Move an MDI child window from one MDI parent window to another.
Transform an MDI child window into an MDI parent and vice versa.
Move controls from one window to another.

Clearly this list includes a couple of tricks that a self-respecting application will probably never need to use. Still, they illustra also provides capabilities that extend it. The result is a framework that works the way you would intuitively expect it to work

Note The samples for this chapter include a project called ImpossibleAPI, which shows one of these "broken rules"-a child window that can jump between different MDI parents whenever the user clicks a button.

All of this raises an interesting question. How can a programming model built on the Windows API actually perform feats tha the Windows API with a fair bit of effort. For example, you could appear to change the border style of a window by destroying information from the previous window.

In fact, this is more or less what takes place in .NET. If you examine the control or window handle (the numeric value that id operations. This signifies that, on an operating system level, .NET actually provides you with a new window or control. The c it's hardly an illusion at all (any more than the illusion that .NET web controls can maintain state, or that television shows co

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev

The Control Class

Chapter 2 introduced .NET control classes, and examined their place in the overall architecture of an application. To summa
You create and manipulate controls and forms using .NET classes. The Common Language Runtime recognizes th
You use a control from the .NET class library by creating an instance of the appropriate class, and adding it to the C
You configure controls by setting properties. In addition, you can react to control events in two ways: by creating an

Every .NET control derives from the base class System.Windows.Forms.Control. Depending on the complexity of the contro

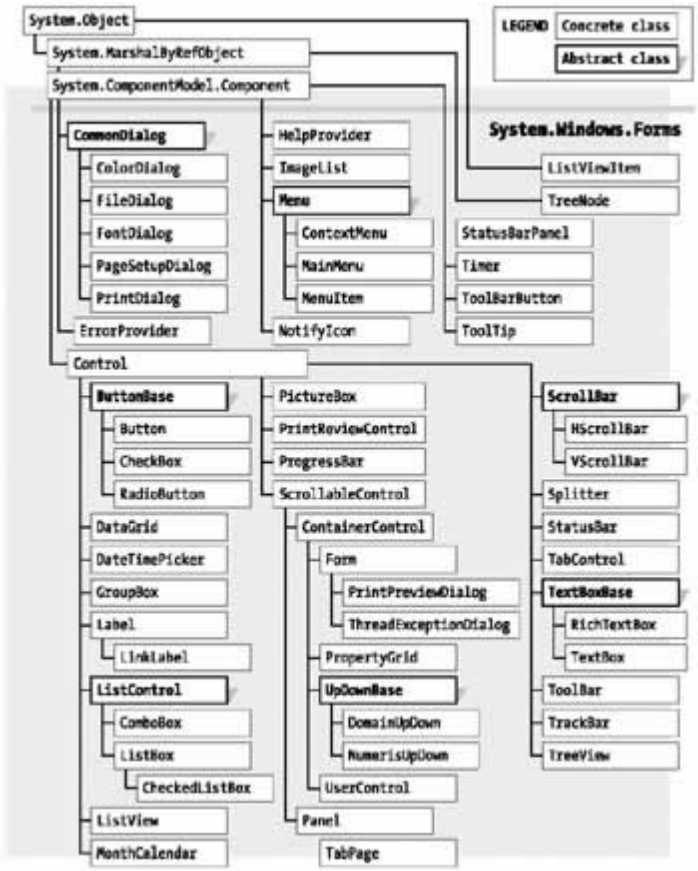


Figure 3-1: Control hierarchy

On its own, the Control class has no real purpose. It's mainly interesting for the basic functionality that it defines. Sorting thro
reset values to their defaults, along with some more useful and unusual members. The sections in this chapter sort through

Table 3-1: Basic Control Members

Member	
Name	

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Position and Size

A control's position is defined by the distance between its top-left corner and the top-left corner of its container. Often, the c measured as the width and height of the control from the top-left point. By convention, the position measurement is positive i

Note In the later chapters on GDI+, you will discover that it is possible to create specialized controls that have irregular boundaries by using the Region property. Chapter 5 previews this technique with irregularly shaped forms.

All values are integers measured in pixels. They are provided through several properties (including Top, Left, Right, and Bott of these properties, the preferred way for setting position is by using the Location property with a Point structure. Similarly, t originate from the System.Drawing namespace.

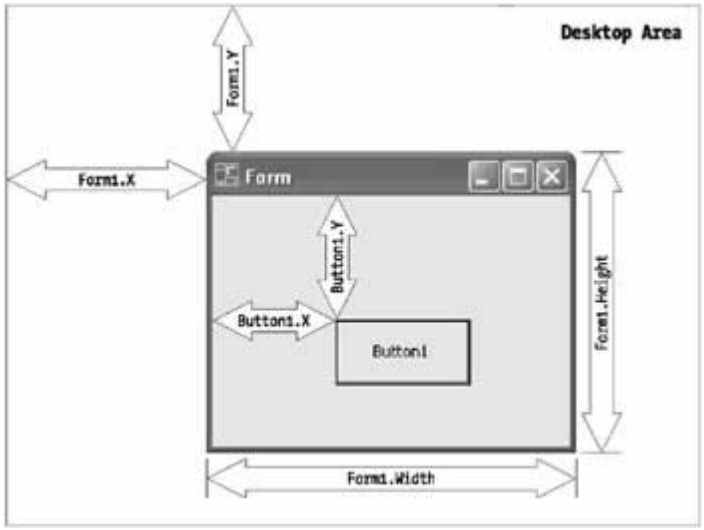


Figure 3-3: Control measurements

The following code shows how you can set the location and size of a control using the Point and Size structures.

```
System.Drawing.Point pt = new System.Drawing.Point();
pt.X = 300;    // The control will be 300 pixels from the left
pt.Y = 500;    // The control will be 500 pixels from the top.
ctrl.Location = pt;

System.Drawing.Size sz = new System.Drawing.Size();
sz.Width = 500;
sz.Height = 60;
ctrl.Size = sz;

// Just for fun, set another control to have the same size.
ctrl2.Size = ctrl.Size;
```

By importing the System.Drawing namespace and using some handy constructors, you can simplify this code considerably

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Color

Every control defines a ForeColor and BackColor property. For different controls, these properties have slightly different mea
background color is the area behind it. These values default to the Windows system-configured settings.

Colors are specified as Color structures from the System.Drawing namespace. It's extremely easy to create a color object, b

- An ARGB (alpha, red, green, blue) color value. You specify each value as integer.
- An environment setting from the current color scheme. You choose the correspondingly named property from the Sy
- A predefined .NET color name. You choose the correspondingly named property from the Color class.
- An HTML color name. You specify this value as a string using the Color-Translator class.
- An OLE color code. You specify this value as an integer (representing a hexa-decimal value) using the ColorTransla
- A Win32 color code. You specify this value as an integer (representing a hexadecimal value) using the ColorTransla

The code listing that follows shows several ways to specify a color using the Color, ColorTranslator, and SystemColors type

```
// Create a color from an ARGB value
int alpha = 255, red = 0;
int green = 255, blue = 0;
ctrl.ForeColor = Color.FromARGB(alpha, red, green, blue);

// Create a color from an environment setting
ctrl.ForeColor = SystemColors.HighlightText;

// Create a color using a .NET name
ctrl.ForeColor = Color.Crimson;

// Create a color from an HTML code
ctrl.ForeColor = ColorTranslator.FromHtml("Blue");

// Create a color from an OLE color code
ctrl.ForeColor = ColorTranslator.FromOle(0xFF00);

// Create a color from a Win32 color code;
ctrl.ForeColor = ColorTranslator.FromWin32(0xA000);
```

The next code snippet shows how you can transform the KnownColors enumeration into an array of strings that represent co

```
String[] colorNames;
colorNames = System.Enum.GetNames(typeof(KnownColor));
```

Changing a color name string back to the appropriate enumerated value is just as easy using the special static Enum.Parse(
chooses the matching one.

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Fonts and Text

The Control object defines a Text property that is used by derived controls for a variety of purposes. For a text box, the Text controls like labels, command buttons, or forms, the Text property refers to static descriptive text displayed as a title or capt

The font of a control's text is defined by the Font property, which uses an instance of the System.Drawing.Font class. Note about the font family, point size, and styles (like bold and italic).

```
// You can create a font with one of the 13 constructors.
ctrl.Font = new Font("Tahoma", 8, FontStyle.Bold);
```

Tip A traditional default font for Windows programs is often Microsoft Sans Serif. However, newer applications since Windows 98 consistently use the slightly more attractive Tahoma font (which is also better for input, as it distinguishes between characters like a lowercase "l" and uppercase "I").You should use the Tahoma font in your applications.

A Control.FontHeight property is also provided, which returns the height of your chosen font in pixels. This setting allows you example, you could manually space lines the appropriate amount when drawing text directly onto a form background.

Note that font families are set using a string, rather than a type-safe enumerated property. If you try to create an object using the Microsoft Sans Serif font. An error does not occur. You may want to explicitly check the Font.Name property to check if

To determine what fonts are installed on the system, you can enumerate through them with the System.Drawing.Text.Install

```
InstalledFontCollection fonts = new InstalledFontCollection();
foreach (FontFamily family in fonts.Families)
{
    lstAvailableFonts.Add(family.Name);
}
```

The online samples for this chapter include a FontViewer utility that uses this technique to create a list of fonts. The user ca window (see Figure 3-5). To perform the font painting, the application uses some of the GDI+ methods you'll see in Chapter 1

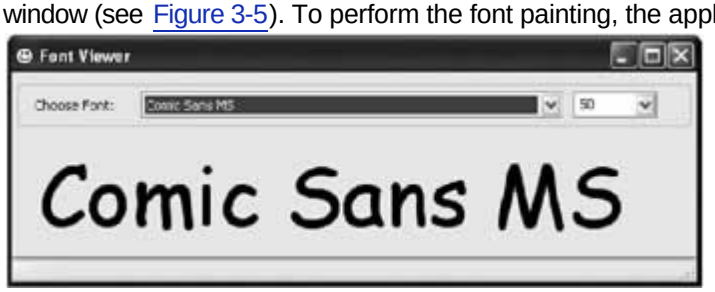


Figure 3-5: A simple font viewer

Note Note that the Font property does not **always** configure the appearance of the text in the control. For example, the font of all windows is defined by the system and is unchangeable. Instead, the Form.Font property defines the default font for all contained controls.

Access Keys

Some controls (namely buttons and menu items) allow a character in their caption to be highlighted and used as an access Ctrl key and that character, the button is "clicked" automatically. To configure these shortcuts keys just add an ampersand ampersand, you'll need to include the text "&&".)

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev

Control Relations

Table 3-2).

Table 3-2: Members for Control Relationships

Member
HasChildren
Controls
Parent
TopLevelControl and FindForm()
Contains()
GetChildAtPoint()
ContextMenu and Menu

Prev

Chapter 3 - Control Class Basics

byMatthew MacDonald
Apress 2002

Companion Web Site

Focus and the Tab Sequence

In the Windows operating system, a user can only work with one control at a time. The control that is currently receiving the example, the button control uses a dotted line around its caption to show that it has the focus. Figure 3-6 shows focused an

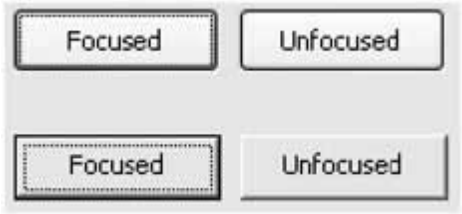


Figure 3-6: Focused buttons

To move the focus, the user can click the mouse or use the tab key. The developer has to take some care to make sure that developer also has to choose the control that should receive the focus when the window is first presented.

All controls that support focusing provide a Boolean TabStop property. When set to true, the control can receive focus. Whe

Tip You should set the TabStop property to false for controls that can accept key presses but are not directly accessed by the user in your application. For example, you might provide a DataGrid control, but use it to display static information. Of course, setting the TabStop to false also means the user will need to use the mouse to scroll the control, if its contents extend beyond the bounds of its display region.

To set the tab order, you configure a control's TabIndex property. The control with a TabIndex of 0 gets the focus first. Wher accept focus. Visual Studio .NET provides a special tool, shown in Figure 3-7, that allows you to quickly set tab order. Just desired order.

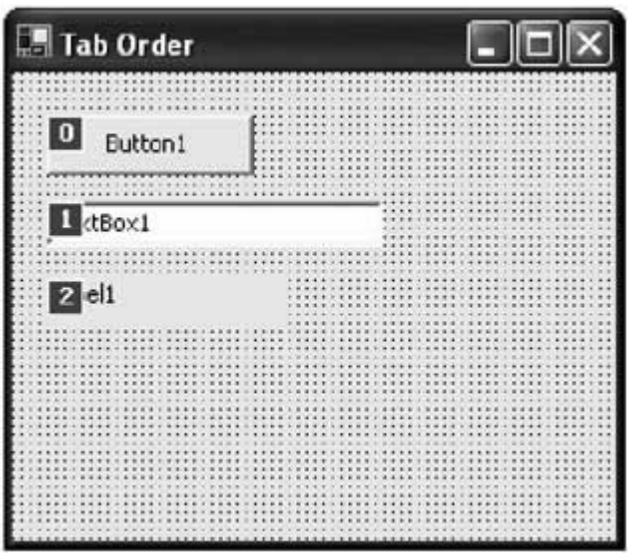


Figure 3-7: The Visual Studio .NET tab order tool

Label controls have a TabIndex setting even though they cannot receive focus. This allows you to use a label with an acces control in the tab order. For that reason, you should give your labels an appropriate place in the tab order, especially if they label's text.)

Controls that are invisible or disabled (commonly known as "greyed out") are generally skipped in the tab order, and are not Visible and Enabled properties, respectively. Note that if you hide or disable a control at design time, the appearance is not time, and it is recommended that custom controls also follow this pattern.

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Responding to the Mouse and Keyboard

Controls also provide some built-in intelligence for dealing with the keyboard and mouse. These include low-level events that information (Table 3-4).

Table 3-4: Events for Reacting to the Keyboard

Event	Description
KeyDown	Occurs when a key is pressed while the current the state of the Alt and Ctrl keys and the key c
KeyPress	This is a higher-level event that occurs once th The event provides a KeyPressEventArgs obje Handled property, which you can set to true to an input control.
KeyUp	This occurs when a key is released, just after t

Generally you will react to the KeyDown and KeyUp events when you need to react to special characters like the arrow keys perform character validation.

If you want to update the display or react to a changed text value in an input control, you would probably not use any of these are made. The Changed event will fire if you modify the text programmatically or the user deletes the text with the right-click

Forms provide a Boolean KeyPreview property. If you set this to true, your form receives key press events when any of its co useful if you are programming a customized interface or a game where you need to take complete control of the keyboard. Y own context-sensitive help system.

Table 3-5: Events for Reacting to the Mouse

Event	Description
MouseEnter	Occurs when the mouse moves into a contro
MouseMove[*]	Occurs when the mouse is moved over a cor current coordinates of the mouse pointer. Be handlers that react to this event can be used
MouseHover	Occurs when the mouse lingers, without mov Usually, you react to this event to highlight th
MouseDown[*]	Occurs when a mouse button is clicked.

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Graphics and Painting

Controls provide a variety of events and properties related to painting (Table 3-6). You'll find information about how to take co

Table 3-6: Graphics-Related Members

Member	Description
BackgroundImage	Allows you to show a picture in the ba
Image (or ImageList and ImageIndex)	These properties aren't a part of the b check boxes, and radio buttons. It allo
ImageAlign	Allows you to align a picture to any si controls.
ResizeRedraw	Indicates whether the control should a
CreateGraphics()	Creates a System.Drawing.Graphics o explained in Chapter 12.
Invalidate() and Refresh()	Triggers a repaint of the control.
Paint event	Occurs when the control is redrawn. P
Resize event	Occurs when the control is resized, ju

You can create an Image object using the static Image.FromFile() method, which reads a standard bitmap format (like a BM

```
button1.Image = Image.FromFile(Application.StartupPath + "\\mypic.bmp");
```

The Image class provides its own set of properties and methods. Some of the most interesting include RotateFlip(), which ch object of the specified size that condenses the information from the original Image.

```
Image myImage, myThumbnail;
myImage = Image.FromFile(Application.StartupPath + "\\mypic.bmp");

// Rotate by 270 degrees and flip about the Y-axis.
myImage.RotateFlip(RotateFlipType.Rotate270FlipY);

// Create a 100 x 100 pixel thumbnail
```

Chapter 3 - Control Class Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Low-Level Members

The .NET framework hides the low-level ugliness of the Windows API, but it doesn't render it inaccessible. This is a major ad capabilities.

For example, if you want to use a DLL or Windows API function that requires a window handle, you can just retrieve the cont immediately before you use it. Changing some properties can cause the control to be recreated, and receive a new handle.

You've probably also realized by now that low-level Windows messages are abstracted away in .NET controls, and replaced that doesn't have a corresponding event, you can handle it directly by overriding the PreProcessMessage() method. (You ca Application.AddMessageFilter() method).

This book focuses on pure .NET programming, and won't get into most of these tricks, which are really workarounds based members described in Table 3-7.

Table 3-7: Low-Level Members

Member	Description
Handle	Provides an IntPtr structure (a 32-bit i
RecreatingHandle	Set to true while the control is being r
GetStyle() and SetStyle()	Sets or gets a control style bit. Gener
PreProcessMessage(), cessCmdKey(), ProcessKeyMessage(), ProcessKeyPreview(), and WndProc()	These methods are involved in proces these methods and use them to proc

Chapter 3 - Control Class Basics

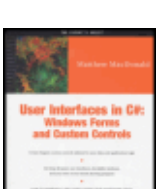
byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter provided a lightning tour through the basics of .NET controls, including how they interact, receive messages, pr
the System.Drawing namespace for creating and managing colors, fonts, images, and more. The next chapter puts this high



Chapter 4 - Classic Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 4: Classic Controls

This chapter considers some of the most common types of controls, like menus, text boxes, and buttons. Many of these cor controls, you will learn about the hyperlink label, checked list box, and rich date controls. I'll also describe menus in detail, a

The final part of this chapter demonstrates two advanced .NET features. First, you learn how you can add support for drag-an

Types of Controls

Not all controls are created equal. .NET uses specialized controls that don't appear on forms, enhance other controls, and p

Invisible Controls

Invisible controls don't require a portion of form real estate. These include controls that never have a visual appearance, like t

When you drag an invisible control onto the form surface, a special icon appears for it in the component tray (see Figure 4-1 added to the form's Controls collection.

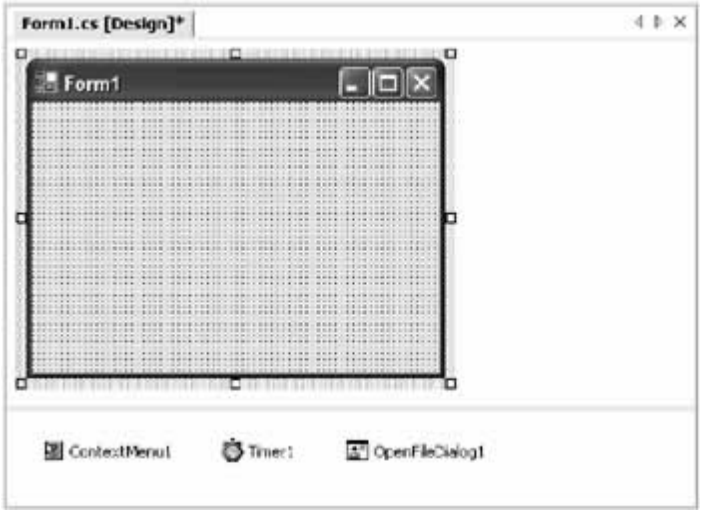


Figure 4-1: The component tray

In some cases, it's worth asking whether invisible controls are really controls at all. For example, the Timer is just a special v marshal calls when interacting with a control. (In other words, there is no need to use the Control.Invoke() method, because

Note Invisible controls don't derive from the Control class. Instead, they derive from System.ComponentModel.Component (the Control class also derives from this class). The Component class adds the basic features needed for an item to be hosted in a container and provides a Dispose() method that causes it to release its resources immediately.

Often, invisible controls can be created more flexibly at runtime in your own code. In some cases (like when you want to sha example of invisible controls, and are described later in this chapter.

Provider Controls

Providers are a special type of invisible control. They extend the properties of other controls on the current form. For example

At design-time, select the appropriate control and look in the Properties window for the property "ToolTip on tipProvid
At runtime, use the ToolTipProvider.SetToolTip() method. You can also use the GetToolTip() method to retrieve a co

Chapter 4 - Classic Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Classic Control Gallery

Now that you've learned about control fundamentals, it's time to look at some of the familiar controls every programmer know

Labels

Label controls are used to place static text on a form. The text is contained in Text property, and aligned according the Text

Table 4-2: Label Properties

Property	Description
RenderTransparent	When set to true, makes the label's b boxes. Of course, you can't control it other situations where you need prof
AutoSize, PreferredHeight, and PreferredWidth	.NET labels support automatic resizi features, set the AutoSize property to
Image and ImageAlign	Label controls can also contain a pict finer-grained control by using separa
UseMnemonic	When set to true, ampersands in the access key, and the focus is forward

LinkLabel

This specialty label inherits from the Label class, but adds some properties that make it particularly well suited to represent

The LinkLabel handles the details of displaying a portion of its text as a hyperlink. This portion is identified in the LinkArea p in the link. Depending on the LinkBehavior property, this linked text may always be underlined, displayed as normal, or it ma

Here's the basic code that creates a link on the web site address:

```
lnkWebSite.Text = "See www.prosetech.com for more information.";

// Starts at position 4, and 17 characters long.
lnkWebSite.LinkArea = new LinkArea(4, 17);
lnkWebSite.LinkBehavior = LinkBehavior.HoverUnderline;
```

You need to handle the actual LinkClicked event to make the link functional. In this event handler, you should set the LinkVis you might start Internet Explorer with the following code:

Chapter 4 - Classic Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Date Controls

Retrieving date information is a common task. For example, requiring a date range is a good way to limit database searches. text boxes that required a specific format of month, date, and year values.

The modern date controls make life much easier. For one thing, they allow dates to be chosen from a graphical calendar vie example). They also allow dates to be displayed in a range of formats.

There are two date controls: DateTimePicker and MonthCalendar. DateTimePicker is ideal for choosing a single date value, drop-down button, a full month calendar page is shown. The user can page from month to month (and even year to year) loo automatically.

The MonthCalendar shows a similar expanded display, with a single month at a time. Unlike the DateTimePicker, it allows t

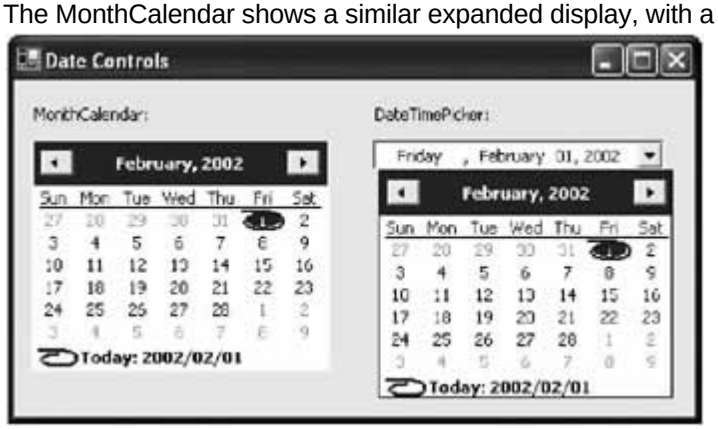


Figure 4-8: The date controls

The DateTimePicker

The DateTimePicker allows a user to choose a single date. One nice thing about the DateTimePicker is that it automatically DateTimePicker.Format property, and the date might be rendered as yyyy/mm/dd format or dd/mm/yyyy depending on the CustomFormat property, and make sure the date is always presented in the same way on all computers. Figure 4-9 shows t

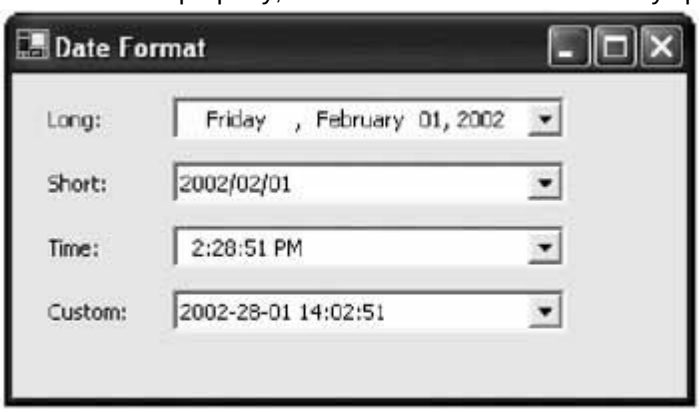


Figure 4-9: Common date formats

The selected date is provided in the Value property. One important detail about date controls is that they always use the Sy configure a date control to show only the day or time portion. In this case, you may need to be careful to retrieve just the ap

For example, imagine you are using a DateTimePicker control, which allows the user to choose the start date for a database information.

Chapter 4 - Classic Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Menus

Applications use two kinds of menus: main and context. Main menus provide a comprehensive set of options just below a w
up" menus that provide additional options. Typically, context menus appear when the user right-clicks a user interface elem
menus, but they tend to be much simpler.

Consider the partial menu shown in Figure 4-11. In .NET, this menu is modeled as a collection of objects, as shown in Figur

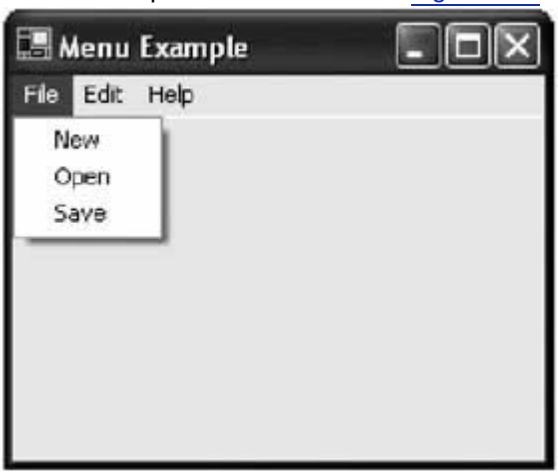


Figure 4-11: A sample menu

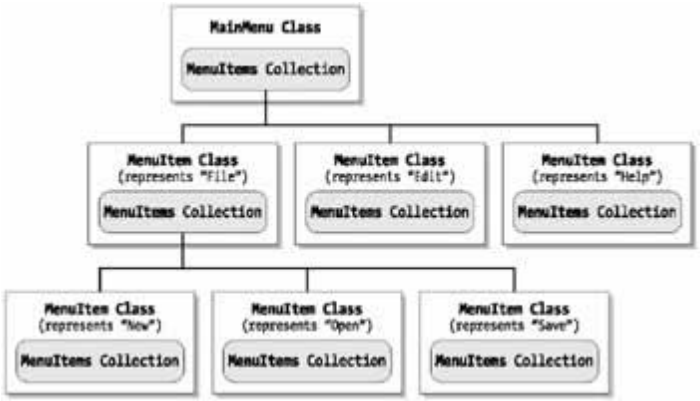


Figure 4-12: Menu objects in .NET

The next code snippet shows how to create the first part of this menu (all the entries contained under the File heading). Gen
the custom menu designer.

```
// Build the menu starting with the deepest nested level
// (in this case, New, Open, and Save).

MenuItem mnuNew = new MenuItem();
mnuNew.Text = "New";

MenuItem mnuOpen = new MenuItem();
mnuOpen.Text = "Open";
```

Chapter 4 - Classic Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Owner-Drawn Menus

If this is your first look at .NET menus, you may be disappointed to see that they don't support the common Windows conve you could purchase a third-party control, but luckily the logic is easy to implement on your own. All it requires is a dash of G

The steps for creating an owner-drawn menu are as follows.

1. Set the OwnerDraw property for the menu items to true. Note that even when you do this, you can still see the menu
2. Handle the MeasureItem event. This is where you tell .NET how much space you need to display the menu item.
3. Handle the DrawItem event. This is where you actually write the output for the item to the screen. In our case, this c

The first step is easy enough. The second step requires a little bit more work. To specify the required size, you need to set t However, the size required depends a great deal on the font and text you use. Fortunately, the MeasureItemEventArgs class method. This method returns a Size structure that indicates the space required.

To keep your code manageable, you should use a single event handler to measure all the menu items. In our example, this

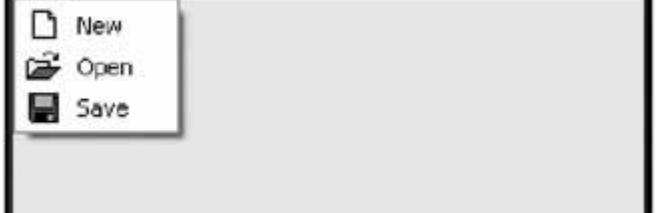
```
private void mnu_MeasureItem(object sender,
 System.Windows.Forms.MeasureItemEventArgs e)
{
    // Retrieve current item.
    MenuItem mnuItem = (MenuItem)sender;

    Font menuFont = new Font("Tahoma", 8);

    // Measure size needed to display text.
    // We add 30 pixels to the width to allow a generous spacing for the image.
    e.ItemHeight = (int)e.Graphics.MeasureString(mnuItem.Text, menuFont).Height
                   + 5;
    e.ItemWidth = (int)e.Graphics.MeasureString(mnuItem.Text, menuFont).Width
                   + 30;
}
```

When displaying a drop-down menu, Windows automatically measures the size of each item, and uses the greatest require

The final step of displaying the menu item is similarly straightforward. You must find the appropriate picture, and write both th the DrawString() and DrawImage() methods. The result is shown in Figure 4-16.



Chapter 4 - Classic Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Drag-and-Drop

Drag-and-drop operations aren't quite as common today as they were a few years ago, because programmers have gradually technique that many users find difficult to master). For example, a drawing program is likely to use a two-step operation (sele drag-and-drop often use it as a shortcut for advanced users, rather than a standard way of working.

Drag-and-drop is also sometimes confused with the ability to "drag" a picture or piece of user interface around a window. This application developed in 13), but it needs to be coded manually. In this section, you will learn about both types of dragging o

"Fake" Drag-and-Drop

True drag-and-drop is a user-initiated way to exchange information between two controls. You don't need to use drag-and-dro the following program that allows a user to click on a picture box, drag it around, and release it somewhere else on the form.

Figure 4-18: Dragging a control around

Conceptually, a control is being dragged and dropped, but all the logic takes place in the appropriate mouse handling events

```
// Keep track of when fake "drag and drop" mode is enabled.
private bool isDragging = false;

// Store the location where the user clicked the control.
private int clickOffsetX, clickOffsetY;

// Start dragging.
private void lblDragger_MouseDown(System.Object sender,
 System.Windows.Forms.MouseEventArgs e)
{
    isDragging = true;
    clickOffsetX = e.X;
    clickOffsetY = e.Y;
}
```

Chapter 4 - Classic Controls

byMatthew MacDonald

2002

Companion Web Site

Validation

The best possible way to prevent invalid input is to make it impossible for users to enter it. You accomplish this by forcing u the fact. If you take this approach, it's important that you report the error as soon as possible, preferably before the user con

Validation events were designed to let you check information as soon as it is entered, rather than waiting for the whole form
Without it, users might be afraid to submit a form because they know there is a possible error.
Users might enter several pieces of invalid data at the same time. If you don't check the data until the form is submi
By the time users submit a form, they might have already forgotten about the particular field they entered incorrectly

Validation solves this information by checking the field as soon as the user is finished entering it and changes focus to anot

In the past, developers have tried to create "do-it-yourself" validation by responding to a control's LostFocus event. The prob each other, trying to move the focus somewhere else.

.NET handles this problem with the Validating and Validated events. These events occur after the user has chosen to move t

1. Leave
2. Validating
3. Validated
4. LostFocus

The Validated event allows you to respond to correctly entered data. The Validating event is more useful. It allows you to ve

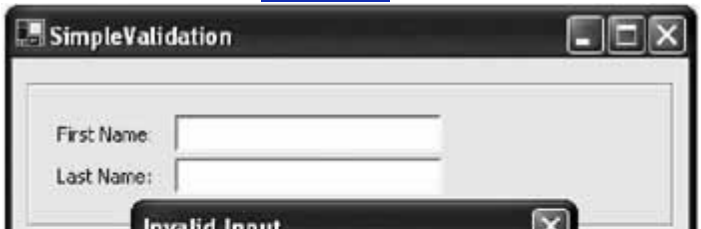
Validation only takes place if the source control (the control to be validated) has the CausesValidaton property set to true. Ir

Table 4-15: .NET Validation

Source Control
CausesValidation is false
CausesValidation is true
CausesValidation is true

A Validation Example

The program shown in Figure 4-20 uses validation to verify that neither text box is left blank. If the user tries to change focus



Chapter 4 - Classic Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter has toured through the most common Windows controls, and demonstrated a few .NET twists like owner-drawn techniques you can use for drag-and-drop support, and the best ways to integrate validation code into your forms. The next

Chapter 5 - Forms

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev

Chapter 5: Forms

Windows are the basic ingredients in any desktop application-so basic that the operating system itself is named after them.

This chapter explores how forms interact and take ownership of one another, and how forms are used to manage events. It a
will amaze your programming colleagues. Finally, the end of this chapter considers the advantages and limitations of visual i

The Form Class

The Form class is a special type of control that represents a complete window. It almost always contains other controls. The

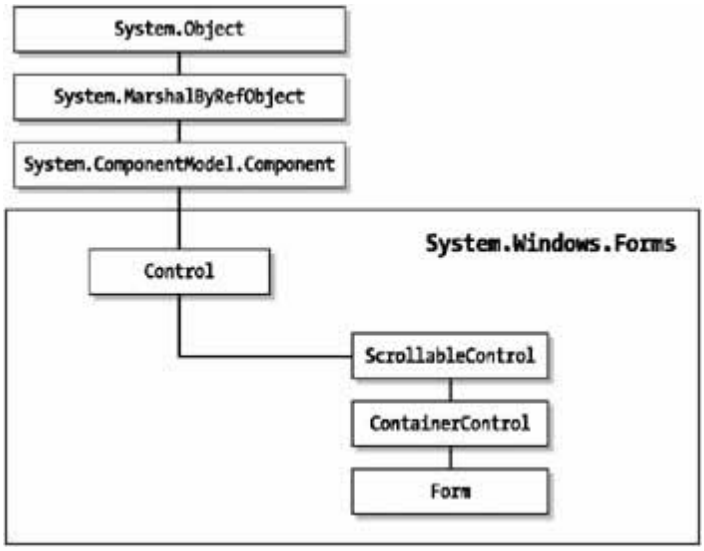


Figure 5-1: The Form class lineage

The Form class provides a number of basic properties that determine appearance and window style. Many of these properties

Table 5-1: Basic Style Properties

Member
FormBorderStyle
ControlBox
MaximizeBox
MinimizeBox
HelpButton
Icon

Chapter 5 - Forms

byMatthew MacDonald

Apress 2002

Companion Web Site

Resizable Forms

Every week, Windows developers from every programming language spend countless hours trying to solve the problem of re
resizable wonders automatically. These components are easy to use, but generally provide mediocre results that aren't suita
fixed-size dialog boxes, making them seem unmistakably primitive. Most developers eventually give in, and write lengthy co

In .NET the picture has finally improved. Two new features-anchoring and docking-provide built-in support for resizable forms
themselves automatically. The catch? It's extremely easy to create the equivalent of the mediocre ActiveX resizing control. I
way with far less effort than was needed before.

Matching a good resizing approach with a sophisticated interface is possible, but it requires a little more subtlety and a few
property. Along the way, you learn how to create scrollable windows and controls, and see a full-fledged Explorer-style appl

The Problem of Size

The resizable forms dilemma stems from the fact that the Windows operating system supports a variety of monitors at seve
on another, or even stretch beyond the bounds of the desktop, obscuring important controls.

For many simple applications, these types of problems are not serious because programmers usually design their applicatio
also generally accepted that users with much larger viewable areas expect to run several programs at once, and purchased
viewable area with larger fonts or extra white space in a dialog box.

However, a document-based application can't afford to ignore these considerations. Users with more available space expect
example Figure 5-11) are irredeemably frustrating.

Insufficient Space
a MIDI-only project), you don't need to use .bun files. wrk
files contain all the necessary project settings and MIDI data
in this case.
Explanation of DirectShow Audio Options
Information on several new audio options in the 'DirectShow
Audio' dialog (Options-Audio) was left out of the User's
Guide and online Help. Below is an explanation of each new
option.

Figure 5-11: A frustrating window

Tip Generally, a form needs to be resizable as soon as it stores more content than it can show at once. For example, a configuration window with a few settings does not need to be resizable, because all the information is available. On the other hand, a window that shows a scrolled text view or a data grid can benefit from more space, and so should be resizable.

Ever since the dawn of Windows 95 and 32-bit programming, developers have found it increasingly difficult to avoid writing m
The emphasis on MDI interfaces has also contributed to this change. In an MDI (Multiple Document Interface) application, u

One common solution is to write procedures that dynamically resize the window, by responding to a resize event or messag
and ugly. It's also hard to alter or debug when the form changes even slightly, and a nightmarish example of hard-coded valu

Minimum and Maximum Size

Chapter 5 - Forms

byMatthew MacDonald

Apress 2002

Companion Web Site

Splitting Windows

One of the most recognizable user interface styles in applications today is the split window look (arguably popularized by Wi paradigm of MDI, and Microsoft has led the change (although many developers, myself included, still favor MDI design for ma

In .NET, split-window designs are refreshingly easy to create, which is a dramatic departure from most other programming e

1. Start by docking a control to one side of the window.
2. Add the Splitter control, and dock it to the same side.
3. Add another control, and set its docking to Fill so it occupies the rest of the window.

The example in Figure 5-19 shows this technique with a TreeView and a ListView. By moving the position of the splitter bar a

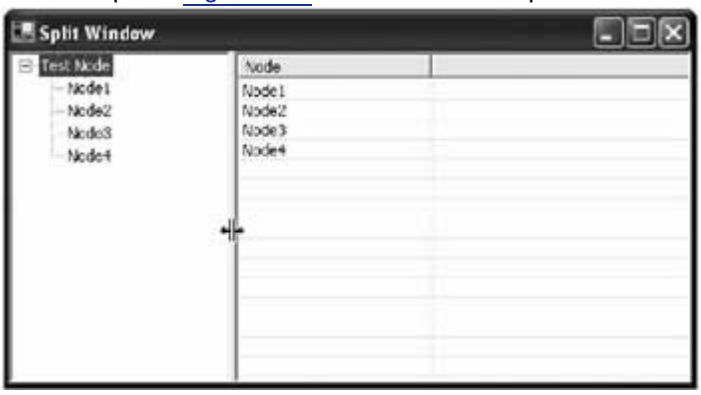


Figure 5-19: A basic splitter bar

In this case, the window is somewhat claustrophobic. To improve the spacing, you can set a buffer using the form's DockPad that, you need to anchor the ListView and TreeView in separate docked Panels, as you consider in the next example.

You can also set the SplitterBar.MinSize property to configure the smallest size (in pixels) to which the first docked control controls on the other side of the splitter.

Docking with Panels

Rather than docking specific controls, as in the preceding example, it's more common to dock two Panel controls, one on ea their spacing. You can then use anchoring to cause the controls inside the Panel to resize themselves to fit its contents. Thi

Figure 5-20 shows an example taken from Chapter 9, which uses a customized TreeView/ListView explorer.

Embedding Control Data
Communications
Deception
Travel
Protection
Multi-Purpose Towelette
Pocket Protector Rocket Pack
Ultra Violet Attack Defender
Guard Dog Pacifier
Cocktail Party Pal
Bullet Proof Facial Tissue
Munitions
Tools
General
ID: 370
Name: Multi-Purpose Towelette
Don't leave home without your monogrammed towelette! Made from lightweight, quick-dry fabric, this piece of equipment has more uses in a spy's day than a Swiss Army knife. The perfect all-around tool while undercover in the locker room: serves as towel, shield, disguise, sled, defensive weapon, whip and emergency food source. Handy bail gear for the Toaster Boat. Monogram included with purchase price.
Close

Chapter 5 - Forms

byMatthew MacDonald

Apress 2002

Companion Web Site

Irregularly Shaped Forms

Irregularly shaped forms are often the trademark of cutting-edge consumer applications like photo editors, movie makers, an shaped form is almost effortless, thanks to the new GDI+ model that's included as part of the .NET framework.

To create a simple shaped window in .NET, all you need to do is create a form, and assign a new shape to its Region prope create a Region, but the easiest is by using the GraphicsPath class, which allows you to build up a complex shape out of a

First, begin by importing the required GDI+ namespace:

```
using System.Drawing.Drawing2D;
```

You can then create and apply the GraphicsPath. The following example code defines an ellipse with the same bounds of th ellipse is displayed (see Figure 5-25).

Figure 5-25: A shaped form

```
private void ShapedForm_Load(object sender, System.EventArgs e)
{
    GraphicsPath path = new GraphicsPath();
    path.AddEllipse(0, 0, this.Width, this.Height);
    this.Region = new Region(path);
}
```

You can see part of the original window border at the top and bottom of the shaped form, and the single contained button in portion that the original rectangular form occupied (that is, just above the left edge of the ellipse), you will not select the form step forward-with ordinary GDI code, shaped controls looked like shapes but still behaved like the original rectangles in man

You can also create a shaped form made up of a combination of shapes. In fact, these shapes don't even need to overlap! Th

```
private void ShapedForm_Load(object sender, System.EventArgs e)
{
    GraphicsPath path = new GraphicsPath();
```

Chapter 5 - Forms

byMatthew MacDonald

Apress 2002

Companion Web Site

Visual Inheritance

Visual inheritance is a fancy name for form inheritance, which is similar to the custom control pattern you use to create spec
things:

Use a common form template (visual appearance) for several different windows. This might be useful to create a wiza
Use form functionality in several different windows. This allows you to create a framework that you might use for diffe
some of the same buttons to close the window or open a file.

And, as with any type of inheritance, visual inheritance gives you many different ways to customize how the descendent cla

To create a simple example of form inheritance, you might create a wizard form like the one shown in Figure 5-30. It uses a
bottom. In this example (found in the code for this chapter under the project name VisualInheritance), the base form is name

Figure 5-30: An ancestor form for a wizard

To create an inherited form that uses this form, you first need to compile the project (unless the form is already stored in a s
choose Add ? Inherited Form. You'll be prompted to choose a new form name, and select the form you want to derive from (s

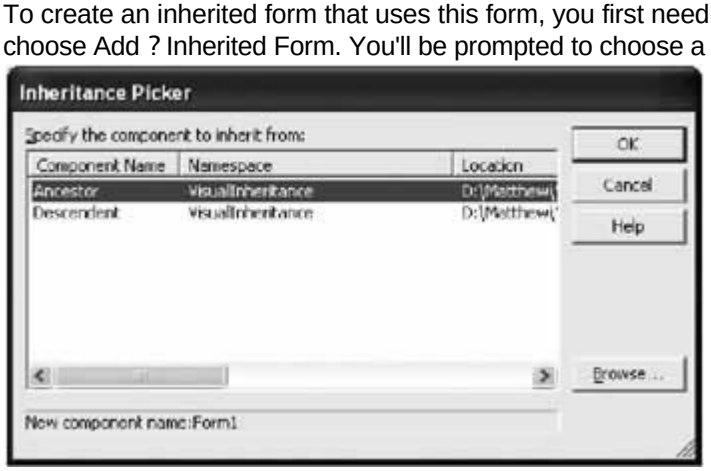


Figure 5-31: Inheriting from a form

Of course, you don't actually need to use the wizard to create an inherited form. All you really need to do is create a Form c

Chapter 5 - Forms

byMatthew MacDonald
Apress 2002

Companion Web Site

The Last Word

In this chapter you've traveled from the basics of Windows forms-creating them, displaying them, and handling their interacti
developments hint at two of the themes I take up in the rest of the book. Shaped forms give just a taste of what you encounte
control designs you explore with control inheritance throughout this book. Both techniques represent the step from ordinary u

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 6: Modern Controls

Many of the controls you've looked at so far (like buttons, menus, and text boxes) have been around since the early days of V organizing complex information. At the same time, the ToolBar and StatusBar revamped the look of the standard Windows a

In this chapter, you learn about all these controls. More important, you learn the tricks and techniques you need to master th synchronize themselves with other controls.

The ImageList

The ImageList is a special type of collection that holds images of a preset size and color depth. Other controls access pictur

Note In some respects, the ImageList isn't really a control. It doesn't have a graphical representation, and the end user never interacts with it directly. On the other hand, ImageList objects are usually created and configured at design time when you are building the user interface. They are also closely linked to other modern controls like ListView, TreeView, and ToolBar controls.

To create an ImageList at design time, drag it onto your form (it will appear in the component tray). The basic properties for t

Table 6-1: ImageList Members

Member
ColorDepth
Images
ImageSize
TransparentColor
Draw()

Tip Transparent regions are a must when mixing custom images and standard controls. If you simply use an icon with a grey background, your interface becomes garish and ugly on a computer where the default color scheme is not used, as a grey box appears around the image.You also run into problems if the icon can be selected, at which point it is highlighted with a blue background.

You can add, remove, and rearrange images using the ListView designer. Just click the ellipsis (…) next to the Images prop Figure 6-1).

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

ListView and TreeView

The ListView and TreeView are probably the most widespread and distinctive controls in modern application design. As far a combine different types of functionality and information. Thanks to Windows Explorer, they are also widely recognized and u

These days, it's hard to find programs that *don't* use TreeView and ListView. The Windows operating system makes heavy u snap-in model follows suit, like SQL Server and even configuration utilities for the .NET platform (see Figure 6-3).

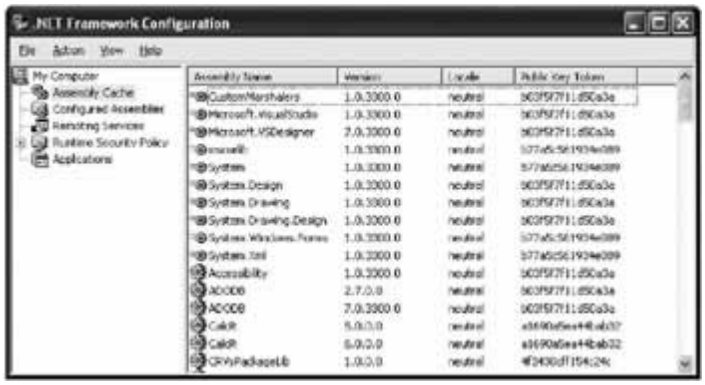


Figure 6-3: Configuring assembly settings in .NET

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Basic ListView

The ListView control is most often used for a multicolumn list of items. It actually supports four distinct modes that you have
LargeIcon, which displays full-sized (32 × 32 pixel) icons with a title beneath each one. Items are displayed from le
SmallIcon, which displays small (16 × 16 pixel) icons with descriptive text at the right. Items are displayed from lef
List, which displays small icons with descriptive text at the right. It's the same as SmallIcon, except it fills items fron
Details, which displays the familiar multicolumn layout. Each item appears on a separate line, and the leftmost colu
information per item.

Tip In Visual Studio.NET, the ListView control uses a designer that allows you to add items and subitems. To use it, just click the ellipses (…) next to the Items property in the Property Window. It's quite impressive, but I won't discuss it in this chapter, as pure code gives a clearer understanding of the issues involved.

To understand the different styles of ListView, it helps to create a simple example. First, create a ListView and two ImageLis

```
listAuthors.SmallImageList = imagesSmall;
listAuthors.LargeImageList = imagesLarge;
```

Once the ImageList is associated, images can be assigned to individual list items by setting a convenient ImageIndex prope

What follows is the code needed to load information into a ListView, in response to a button click. This example relies on a C
databases and .NET), although you can look at the online code for this chapter to see the details. As with most of the examp

```
private void cmdFillList_Click(object sender, System.EventArgs e)
{
    // Fill a DataTable using a helper class (not shown).
    DataTable dt = StoreDB.GetProducts();

    foreach (DataRow dr in dt.Rows)
    {
       // Create the item, with the text from the ModelName field.
       ListViewItem listItem = new ListViewItem(dr["ModelName"].ToString());

       // Give every item the same picture.
       listItem.ImageIndex = 0;

       // Add the item to the ListView.
       listAuthors.Items.Add(listItem);
    }
}
```

This is ListView code is at its simplest. ListViewItem objects are created, and added to the list. The ListViewItem constructo
must use the exact same ordering. The appropriate picture is chosen based on the view style.

Chapter 6 - Modern Controls

byMatthew MacDonald
Apress 2002

Companion Web Site

Advanced ListView Tricks

To unlock all the functionality of the ListView control, you need to delve deeper into the .NET classes that support it. Some c

Table 6-3: Advanced ListView Members

Member	Description
Activation and HoverSelection	Activation determines how items in when it hovers over an item. The Ho over it. This formerly cutting-edge fe
Alignment	Sets the side of the ListView that ite
AllowColumnReorder	When set to true, the user can drag code.
AutoArrange and ArrangeIcons()	In SmallIcon and LargeIcon view, th anywhere by the user.
CheckBoxes, CheckedIndices, and CheckedItems	When CheckBoxes is true, every ite ListViewItem.Checked property of e values (CheckedIndices).
FullRowSelect	When set to true, the entire row will setting for database applications tha
GridLines	Displays attractive column and row information.
HeaderStyle	Allows you to configure whether col
LabelEdit	When set to true, ListViewItem text
LabelWrap	Allows the text label to wrap in one
Sorting	Allows you to specify an ascending subitems).
BeginUpdate() and EndUpdate()	Allows you to temporarily suspend
EnsureVisible()	Scrolls to make sure a specified Lis
GetItemAt()	Retrieves the ListViewItem at the giv
AfterLabelEdit, and BeforeLabelEdit events	Events that fire before and after a la that allows you to cancel the edit

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Basic TreeView

The TreeView is a hierarchical collection of elements, which are called nodes. This collection is provided through the TreeVi

```
treeFood.Nodes.Add("Apple");
treeFood.Nodes.Add("Peach");
treeFood.Nodes.Add("Tofu");
```

In this example, three nodes are added with descriptive text. If you've worked with the TreeView before through its ActiveX c doesn't require a unique key for relating parent nodes to child nodes. This means it's easier to quickly insert a new node. It able to distinguish duplicates. For example, the only difference between the two "Apple" entries in the example is their resp

To specify more information about a node, you have to construct a TreeNode object separately, and then add it to the list. In

```
TreeNode newNode = new TreeNode();
newNode.Text = "Apple";
newNode.Tag = 1;
treeFood.Nodes.Add(newNode);
```

In this case, a simple integer is used, but the Tag property can hold any type of object if needed, even a reference to a corre

```
foreach (DataRow drFood in dtFoods.Rows)
{
    TreeMode newNode = new TreeNode();
    newNode.Text = drFoods["Name"].ToString();
    newNode.Tag = drFood;
    treeFood.Nodes.Add(newNode);
}
```

TreeView Structure

Nodes can be nested in a complex structure with a virtually unlimited number of layers. Adding subnodes is similar to adding collection.

```
TreeNode node;

node = treeFood.Nodes.Add("Fruits");
node.Nodes.Add("Apple");
node.Nodes.Add("Peach");

node = treeFood.Nodes.Add("Vegetables");
node.Nodes.Add("Tomato");
```

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Advanced TreeView Tricks

The TreeView is a sophisticated control, and it provides a great deal of customization possibilities. Some of the additional ap

Table 6-7: TreeView Appearance Properties

Property	Description
CheckBoxes	Set this to true to display a check
FullRowSelect	When set to true, selecting a node
HotTracking	When set to true, the text in a nod
Indent	Specifies the left-to-right distance
ShowLines, ShowPlusMinus, and ShowRootLines	Boolean properties that configure t expand a node, and the root lines
Sorted	When set to true, nodes are sorte sort order, other than to add the no

The TreeNode also provides some useful properties that haven't been discussed yet (Table 6-8). Mainly, these properties allo and foreground color, and determine its relatives, as you saw earlier.

Table 6-8: TreeNode State Properties

Property	Description
Checked	True if you are using a TreeView with check b
IsEditing	True if the user is currently editing this node's
IsExpanded	True if this node is expanded, meaning its chi
IsSelected	True if this is the currently selected node. Onl TreeView.SelectedNode property.
IsVisible	True if the node is currently visible. A node is programmatically show a node, use its Ensure

Chapter 6 - Modern Controls

byMatthew MacDonald
Apress 2002

Companion Web Site

Taming the TreeView

The TreeView control provides a sophisticated infrastructure that allows it to be used in countless different ways. Each indivi
underlying data it represents. That means that the TreeView is an ideal control for subclassing.

A Project Tree

You can easily create custom TreeView classes that are targeted for a specific type of data. Consider the ProjectTree class

```
public class ProjectTree : TreeView
{
    // Use an enumeration to represent the three types of nodes.
    // Specific numbers correspond to the database field code.
    public enum StatusType
    {
        Unassigned = 101,
        InProgress = 102,
        Closed = 103
    }

    // Store references to the three main node branches.
    private TreeNode nodeUnassigned = new TreeNode("Unassigned", 0, 0);
    private TreeNode nodeInProgress = new TreeNode("In Progress", 1, 1);
    private TreeNode nodeClosed = new TreeNode("Closed", 2, 2);

    // Add the main level of nodes when the control is instantiated.
    public ProjectTree() : base()
    {
        base.Nodes.Add(nodeUnassigned);
        base.Nodes.Add(nodeInProgress);
        base.Nodes.Add(nodeClosed);
    }

    // Provide a specialized method the client can use to add nodes.
    public void AddProject(string name, StatusType status)
    {
        TreeNode nodeNew = new TreeNode(name, 3, 4);
        nodeNew.Tag = status;

        switch (status)
```

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The ToolBar

The ToolBar control represents a strip of buttons that gives access to various features in an application. Conceptually, toolba features in an application.

Toolbar buttons can include text and variable-size pictures. You can layer several toolbars on a form, and attach drop-down

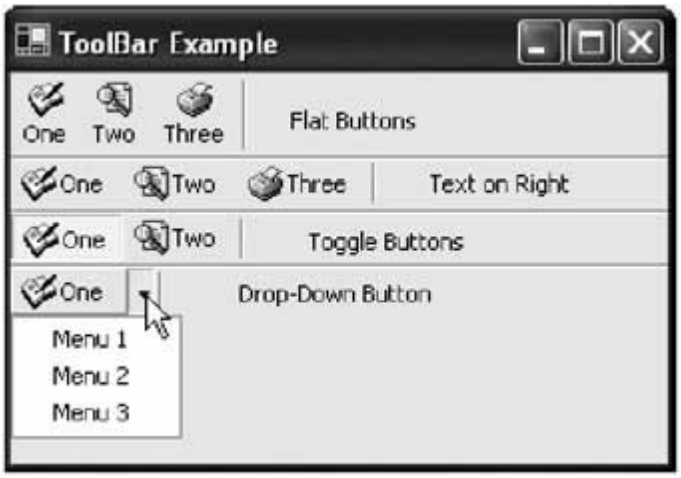


Figure 6-10: Toolbar styles

Unfortunately, .NET does not currently provide any way to create a "cool bar," the snap-in toolbar strip used in applications l then, you can start to create your own using the information in Chapter 8, or use ActiveX interop with the components provid

Most of the ToolBar properties are appearance-related (see Table 6-9).

Table 6-9: Appearance-related ToolBar Properties

Member
Appearance
AutoSize
BorderStyle
ButtonSize
Divider
DropDownArrows
ImageList

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Synchronizing the ToolBar

Usually, a toolbar duplicates functionality that is available in a window's menu. This can result in some rather tiresome state button at the same time. To simplify this process, you can create a customized menu class that automatically forwards its s

This custom control project is a little trickier than some of the previous examples. For one thing, it limits your ability to use th

There's also more than one way to approach this problem. One possibility is to create a custom MenuItem class that stores

```
public class LinkedMenuItem : MenuItem
{
    public ToolBarButton LinkedButton;

    // To save space, only one of the original constructors is used.
    public LinkedMenuItem(string text) : base(text)
    { }

    public bool Enabled
    {
        get
        {
            return base.Enabled;
        }
        set
        {
            base.Enabled = value;
            if (LinkedButton != null)
            {
                LinkedButton.Enabled = value;
            }
        }
    }
}
```

On the downside, this technique requires you to shadow the Enabled property (because it is not overridable), which is a sleig extremely flexible approach. You could even replace LinkedButton with a collection that provides a whole series of controls th programming is used to test the LinkedButton property before attempting to configure the referenced control, in case it has n

The client program can create a linked menu using this class like this:

```
MainMenu menuMain = new MainMenu();
this.Menu = menuMain;
```

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The StatusBar

The StatusBar control is used to display brief information throughout the life of the application. This information should never
it. This information should also be kept to a minimum to prevent a cluttered interface. Some possible status bar information i

Information about the application mode or operating context. For example, if your application can be run by n
user level (e.g., Administrator Mode). Similarly, a financial application might provide a label indicating U.S. Currency
Information about the application status. For example, a database application might start by displaying Ready o
database. This technique avoids stalling advanced users with a confirmation window where they need to click an O
Information about a background process. For example, Microsoft Word provides some information about print o
Information about the current document. For example, most word processors use a status bar to display the cu
display ancillary information like the total number of files in a folder.

These are some of the most useful ways to use a status bar. Status bars should never be used to display the current time.
tray anyway.

Although a status bar can be docked to any side (as set by the Dock property), it is always placed at the bottom of the wind

Chapter 6 - Modern Controls

byMatthew MacDonald
Apress 2002

Companion Web Site

Basic StatusBar

There are two ways to use a status bar. You can display a single piece of text, or a combination of panels that can contain t

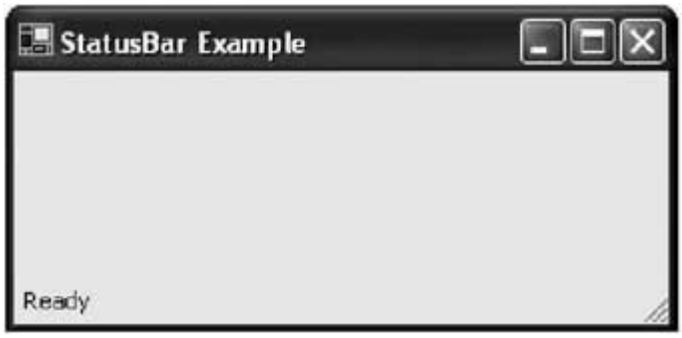


Figure 6-13: The simplest possible status bar

```
statusBar.ShowPanels = false;
statusBar.Text = "Ready";
```

You can also tweak the Font for the current text and the SizingGrip property, which enables or disables the grip lines on the

Alternatively, you can create a status bar that contains different pieces of information. This information is represented as a c

```
StatusBarPanel pnlNew = new StatusBarPanel();
pnlNew.Text = "Ready";
statusBar.Panels.Add(pnlNew);
```

The StatusBarPanel object provides several appearance-related properties (see Table 6-11).

Table 6-11: StatusBarPanel Properties

Property
Alignment
AutoSize
BorderStyle
Icon
MinWidth

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The TabControl

The TabControl is another staple of Windows development-it groups controls into multiple "pages." The technique has becor organized space. It's also easy to use because it recalls the tabbed pages of a binder or notebook. Over the years, the tab c

In .NET, you create a TabControl object, which contains a collection of TabPage objects in the TabPages property. Individua assuming your form contains a TabControl called tabProperties.

```
TabPage pageFile = new TabPage("File Locations");
TabPage pageUser = new TabPage("User Information");

// Add controls to the tab pages.
// The code for creating and configuring the child controls is omitted.
pageUser.Controls.Add(txtFirstName);
pageUser.Controls.Add(txtLastName);
pageUser.Controls.Add(lblFirstName);
pageUser.Controls.Add(lblLastName);
tabProperties.TabPages.Add(pageFile);
tabProperties.TabPages.Add(pageUser);
```

The output for this code is shown in Figure 6-16.

Figure 6-16: The TabPage control

Note Chapter 11 presents an example that allows you to add controls to a TabPage without needing to supply a Location property. Instead, the layout is managed automatically.

The TabControl is easy to work with, and usually configured at design time. Some of its members are described in Table 6-1

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The NotifyIcon

In past programming frameworks, it's been difficult to use a system tray icon. In .NET it's as easy as adding a simple NotifyI

Note Like almost all invisible controls, the NotifyIcon doesn't inherit from the Control class. In other words, the list of members in Table 6-14 is essentially all you have to work with.

Table 6-14: NotifyIcon Members

Member	Description
ContextMenu	The ContextMenu linked the system tray.
Icon	The graphical icon that appears in the sys
Text	The tooltip text that appears above the sy
Visible	Set this to true to show the icon. It defaul
Click, DoubleClick, MouseDown, MouseMove, and MouseUp events	These events work the same as the Cont

The NotifyIcon is an invisible control that appears in the component tray when added at design time. In many cases, it's mo application that loads into the system tray and waits quietly, monitoring for some system event or waiting for user actions. I example of this technique is shown in Chapter 11.

Chapter 6 - Modern Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

In this chapter you've toured some of the most important controls used in typical Windows user interfaces. Even as I've intro control classes. Developing custom controls is one of the key ways to tame these full-featured controls. You'll see more exa reads information from the underlying file system automatically.

Chapter 7 - Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev Next

Chapter 7: Custom Controls

Overview

Custom controls are a key theme in .NET development. They can help your programming style by improving encapsulation, swap out one control and replace it with a completely different one without rewriting your form code). Of course, custom cont modern interface. Generally, developers tackle custom control development for one of three reasons:

To create controls that abstract away unimportant details and are tailored for a specific type of data. You saw this m
To create controls that provide entirely new functionality, or just combine existing UI elements in a unique way. An e
To create controls with a distinct original look, or ones that mimic popular controls in professional applications (like chapter, and returned to in Chapter 13, with GDI+.

Creating custom controls in .NET is far easier than it has been in languages like C++ or VB, where you typically need to emb controls can be difficult to distribute because every version needs to be registered. Creating ActiveX controls also requires a and state management.

In .NET, creating a custom control is as easy as creating an ordinary class. You simply inherit from the best possible ances an existing project, and then decide later to place it in a separate assembly that can be shared with other programmers.

This chapter introduces the different types of custom controls, and the types of problems they can solve. You will look at sev techniques like creating multithreaded controls. However, you won't learn how to make these controls behave happily in Vis the next chapter.

Prev Next

Chapter 7 - Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Types of Custom Controls

Developers often make a distinction between three or four types of controls:

User controls are the simplest type of control. They inherit from the System.Windows.Forms.UserControl class, and (like a group of text boxes for entering address information).

Inherited controls are generally more powerful and flexible. With an inherited control, you choose the existing .NET c adds properties and methods. The examples you've looked at so far in this book, including the custom TreeViews ar

Owner-drawn controls generally use GDI+ drawing routines to generate their interfaces from scratch. Because of thi require the most work and provide the most customizable user interface. You'll see them in Chapter 13.

Additionally, in this chapter you consider extender providers, which aren't necessarily controls at all. These compone user interface.

The distinction above is slightly exaggerated. For example, you can create a user control that uses GDI+ drawing with other you can inherit from one of the intermediary classes to get a different level of support. For example, a control that contains o provide scrolling might inherit from ScrollableControl.

Chapter 7 - Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Control Projects

When designing a custom control, you could create the control class directly in your application project. In this case, you'll n add flexibility, reuse your control, and add design-time support, you need to create a dedicated project for your custom contr assembly that contains the control classes.

The Class Library Project

Typically, you'll create your control as either a Class Library Project (for most custom control projects) or a Windows Contro project type does is configure the default references and the namespaces that are initially imported into your project. The im stand-alone executable. This DLL can then be shared with any other project that needs to use the control. Figure 7-1 shows

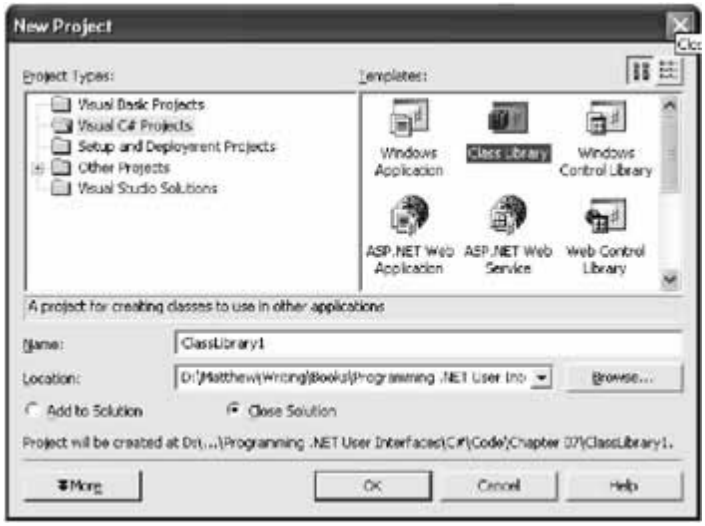


Figure 7-1: Creating a control project

When you begin your control project, you will probably find that you need to add a few assembly references and import som missing reference.

Typically, you need to add references to the System.Windows.Form.dll, System.Drawing.dll, and System.Design assemblie

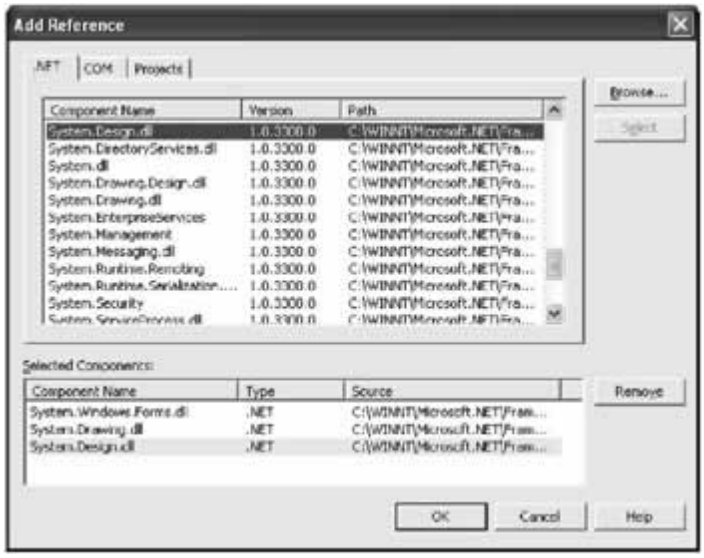


Figure 7-2: Adding required assemblies

Chapter 7 - Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Inherited Controls

Inherited controls are an ideal way to take functionality from the .NET base classes, and extend it. An inherited control can b is filled with examples of inherited controls. For example, LinkLabel derives from Label and CheckedListBox derives from Lis

Unlike user controls, there is no design-time support for creating an inherited control. You simply create a class that derives awkward to use in Visual Studio .NET. For example, it's difficult to add inherited controls to a form except through code. You

Inherited controls are generally more powerful than user controls, and more likely to be used across applications (and even c the reasons that programmers develop inherited controls are to set defaults (for example, a control that automatically configu

So far in this book, you've seen the following examples of inherited controls:

In
In Chapter 4, you saw an inherited menu control that handles its own drawing to allow custom fonts and embedded
In Chapter 5, you saw inherited Form controls with visual inheritance.
In Chapter 6, you saw custom ListView and TreeView examples that support specific types of data.

In this chapter, I'll present two more advanced inherited control examples.

Inherited Controls or User Controls?

So, how do you know when to create a user control, and when you need a fullfledged inherited control? It's not always an ea few pointers that you should consider before embarking on a custom control project:

User controls are easier and faster to program. If you don't anticipate reusing the control frequently in different scena
If your control closely resembles an existing .NET control, it's probably best to create an inherited control. With a us access to the members of the original control.
Inherited controls provide a fine-grained level of reuse. User controls typically provide only a few members, and thus
User controls are well suited if you want to ensure that a block of interface is recreated *exactly* in more than one situ standardized appearance.

If you want to integrate more than one control, you have two choices: you can use composition with a user control, or you ca a TreeView and ListView), but make the links optional. The application programmer can then use them separately or togethe however, the application programmer can only control the size taken by the full user control.

The DirectoryTree Control

The DirectoryTree control inherits from the standard TreeView and adds the features needed to display a hierarchical view o useful.

Perhaps most important, it fills itself by reading subdirectories "just in time." This means that the control operates very quick actually shown. The collapsed branches all have a dummy node inserted. Every time a directory branch is expanded, the inl directories are read from the disk. (You see a variation of this technique to allow efficient data access in Chapter 9).

The full code listing follows. Notice that the currently selected drive is stored as a single character string (technically, a Char currently highlighted directory. That approach would provide better control for the application programmer, but it would compl

```
using System;
using System.IO;
using System.Windows.Forms;
```

Chapter 7 - Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Custom Extender Providers

Extender providers were first introduced in Chapter 4 as interesting components that extend other controls. Examples are th error icon; and the HelpProvider, which invokes context-sensitive Help on a control's behalf when the F1 key is pressed. Pro contemplating a custom provider.

Nonetheless, custom providers can achieve some remarkable tricks. In this section, I demonstrate two extender providers, o Help icon. Both of these classes are found in the ExtenderProviderControls project provided with the online samples. The tes

Tip To create an extender provider, it's easiest to create the custom provider class in a class library project, compile it into a DLL file, and then reference the DLL file from another project by choosing Customize Toolbox. (In fact, this approach is generally the easiest way to integrate inherited controls.) When you add the reference to the extender provider assembly, any associated extender control automatically appears in the ToolBox.

The Menu Text Provider

The MenuTextProvider extends ordinary menus by associating each item with a unique Help string. When the user hovers o interface convention I've mentioned before, and while it's not very useful for the average user, it does provide a good introduct

Choosing a base class

The first step when creating an extender provider is to create a class that implements the IExtenderProvider interface and us interface). This can be any type of class, including a user control, inherited control, or just a basic Component class that doe

A control-based provider, like the MenuTextProvider, uses a dedicated control to display information in a specific location on add the MenuTextProvider to any form, and it will act as an ordinary status bar and update its display to provide the appropri StatusBarPanel class. You could then add it to an existing status bar.

Choosing the object to extend

Once you've decided what type of provider you are creating, your next decision is to determine the type of object that you ar classes. To specify the appropriate type of object, you need to handle the IExtenderProvider.CanExtend() method. In this m extended by your provider. To make this decision you can evaluate any information about the target, including the type (the true if the object can be extended.

The MenuTextProvider only extends the MenuItem object. Here's the code that enforces this restriction:

```
public class MenuTextProvider : StatusBar, IExtenderProvider
{
    public bool CanExtend(object extendee)
    {
        if (extendee.GetType() == typeof(MenuItem))
        {
            return true;
        }
        else
        {
            return false;
```

Chapter 7 - Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter considered one of the most important ingredients in advanced user interfaces: custom controls. You learned ho also learned about inherited controls and the different model they provide.

One topic that hasn't been considered in great detail is Visual Studio .NET's sometimes quirky design-time support of custo develop the designers and type editors that allow them to behave properly in the IDE.

Chapter 8 - Design-Time Support for Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 8: Design-Time Support for Custom Controls

Overview

The Custom Controls you have explored so far are full of promise. Being able to drop a tool like a directory browser or thumb advantage. However, there is one caveat. Though your custom controls perform wonders at runtime, many of them act oddly appear with a generic icon. Inherited controls need to be created in code, unless you create a similar control and manually e

This chapter sorts through these quirks and shows how to create a control that behaves properly at design time. Some of th

- Using attributes to classify and describe properties and other aspects of your control.
- Adding your control to the toolbox-with an icon.
- Using prebuilt UITypeEditor classes that allow directory names, collections, and other property types to be edited a
- Creating your own UITypeEditor classes that allow your control to be configured with custom dialog windows.
- Using control designers to filter out unused properties, add designer wizards, and other tricks.
- Licensing your controls.

This chapter will explore these topics, and use them to revise some of the examples introduced in the previous chapter.

Chapter 8 - Design-Time Support for Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Control Designer Basics

Adding a basic level of design-time support to your controls is easy. In this section you learn how you can outfit your control

Attributes

Designer attributes are the first line in custom control support. They instruct the IDE how to treat various parts of your control syntax), or use some sort of visual tool. With attributes, the information that describes your control can be easily created an

The previous chapter developed a Progress user control that displayed a synchronized label paired with a progress bar. To r

Figure 8-1: Nondescript properties

You can improve on this situation using attributes. The example below adds a Description, Category, and DefaultValue attrib readability.

```
[Description("The current value (between 0 and Maximum) which sets " +
 "the position of the progress bar"), Category("Behavior"), DefaultValue(0)]
public int Value
{
    get
    {
        return Bar.Value;
    }
    set
```

Chapter 8 - Design-Time Support for Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Testing Custom Controls

Testing custom controls can sometimes be a little awkward. Visual Studio .NET provides several options:

You can add test forms directly to your control class projects and remove them when they are no longer needed (or
compiled DLL).
You can create a solution with two projects: one that tests the control, and one that uses the control. However, you
behavior.
You can open two instances of Visual Studio .NET, one with the control project, and one with the control test projec
design-time behavior.
You can use the specialized PropertyGrid control examined in this section.

Even if you don't follow one of these specialized approaches, if an error occurs in your custom control code and the .vb sour
execution on the offending line.

Debugging Design-Time Support

If you test your custom control in a project with the control source code, you are able to set breakpoints and use other debug
control is running "live."

This limitation can hamper your testing to a certain extent. Developing good design-time support for your control requires a
the IDE itself. To accomplish this, add your control to the toolbox, and then configure your custom control project to start Vis
custom designer code that are triggered as the control is manipulated in the IDE.

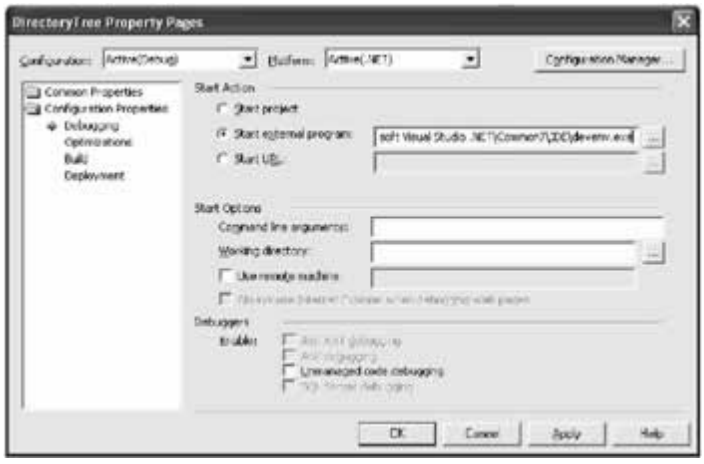
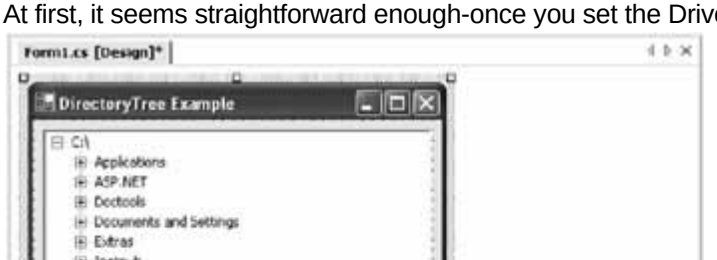


Figure 8-8: Debugging design-time support

Testing for Design Mode

You may also notice some quirky design-time behavior that doesn't occur if the control is added programmatically. For exam

At first, it seems straightforward enough-once you set the Drive property, the corresponding directory tree appears. You can



Chapter 8 - Design-Time Support for Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Custom Designers

One of the problems with traditional ActiveX control development is that details about a control's design-time behavior are mi

A control designer provides the design-time behavior for a control. The .NET framework provides a basic control designer in

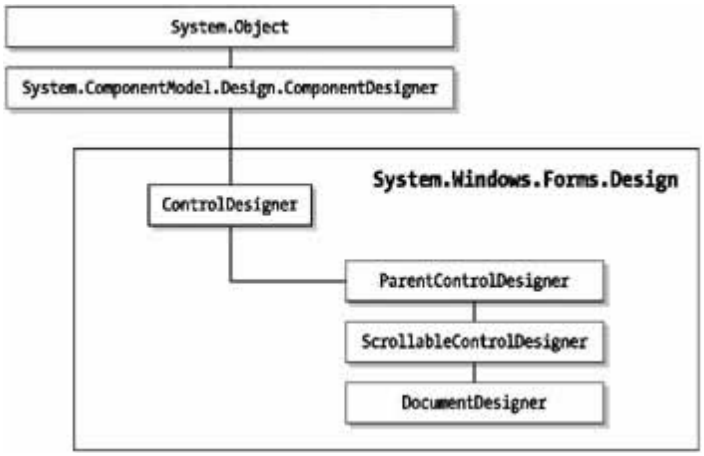


Figure 8-11: Control designer classes

Controls can also derive their own custom designers. Why would you create your own designer?

- To add special designer tools, like context menu options.
- To remove inappropriate events or properties from view (or add designtime only events or properties).
- To add support for controls that contain other controls (like the toolbar) or controls with special needs (like menus).

The next few sections consider all these topics by designing and enhancing a DirectoryTreeDesigner class that works with th

Filtering Properties and Events

Sometimes, an event or property needs to be hidden from a control, but not removed entirely. For example, the ProgressBa Text property should be hidden from the Properties window.

If you are defining or overriding a property, you can use the Browsable attribute to keep it from appearing in the Properties w automatically at runtime based on the Drive property. The TreeView.Nodes property is not overridable, so you can't use the

Designers provide six methods from the IDesignerFilter interface that you can override to filter properties, events, and attribu

Table 8-3: ControlDesigner Filtering Methods

Method
PostFilterAttributes
PostFilterEvents
PostFilterProperties

Chapter 8 - Design-Time Support for Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev Next

Data Types and UITypeEditors

The Properties window does a solid job of providing support for all common data types. Basic data types like strings and nu
Properties window automatically provides a drop-down list with the values from the enumeration.

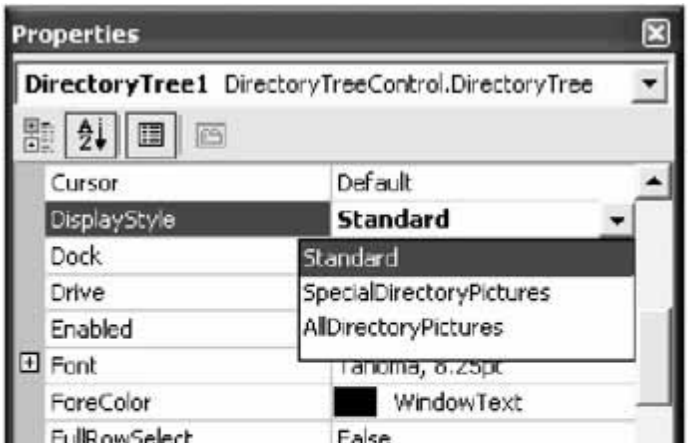
For example, consider the DisplayStyle property shown here:

```
public enum Display
{
    Standard,
    SpecialDirectoryPictures,
    AllDirectoryPictures
}

private Display displayStyle;

public Display DisplayStyle
{
    get
    {
        return displayStyle;
    }
    set
    {
        displayStyle = value;
    }
}
```

The enumerated values are shown in the Properties window (see Figure 8-15).



Chapter 8 - Design-Time Support for Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Licensing Custom Controls

Licensing in the .NET world is far more customizable and far less painful than it was with ActiveX controls. The .NET framew extending these classes, you can grant or allow access to your control, using ordinary .NET code to check external resource

Simple LIC File Licensing

To best understand .NET licensing, it helps to start with a simple example using the LicFileLicenseProvider class. This class you look at next.

The LicFileLicenseProvider searches for a text file in the same directory as the control assembly. This LIC file uses the contr DirectoryTreeControl.DirectoryTree.LIC. Inside this file is a simple predefined text string in the format "[Component] is a licen

Thus, the contents of the DirectoryTreeControl.DirectoryTree.LIC file would be:

```
DirectoryTreeControl.DirectoryTree is a licensed component.
```

This file must be placed in the client project's bin directory (where Visual Studio .NET compiles the final exe just prior to laun

Note It's worth noting that these LIC files don't need to be distributed with a client application.When you compile a Windows program, a license.licx file is created with all the licensing information for all license controls. This file is compiled as a binary resource and embedded in the final client assembly. However, if another developer wants to create a client application with your control, a LIC source file **is** needed.

To enforce LIC file licensing, you need to add a LicenseProvider attribute to your control class that tells .NET to use the LicF

```
[LicenseProvider(typeof(LicFileLicenseProvider))]
 public class DirectoryTree : TreeView
```

Additionally, you need to create the license when the control is created, using the static Validate() method of the LicenseMa

```
private License license;

public DirectoryTree()
{
    license = LicenseManager.Validate(this.GetType(), this);
}
```

The Validate() method throws a LicenseException if it doesn't find the correct string in the LIC file, and refuses to create you

Finally, you need to dispose of the license when the control is disposed.

```
protected override void Dispose(bool disposing)
{
    if (license != null)
```

Chapter 8 - Design-Time Support for Custom Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter covered a lot of ground with custom controls. The story doesn't end here, however. In the coming chapters you everything from data access to rich graphics. In all these cases, knowing how to match custom controls with an appropriate components, it's indispensable.

To see what other developers are creating, you can take your search online and begin working with the latest user interface critique your code creations, but I just might provide a link from this book's web site (http://www.prosetech.com) if there's en

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 9: Data Controls

Overview

It's often remarked that a large percentage of Windows applications are little more than attractive window dressings over a re
responsibility of this type of software is to allow highly structured data entry and provide reportgenerating modules that sumr
concentrated in the user interface tier.

Of course, databases aren't only used for workflow and administrative software. Almost every application needs to connect to
site is really just an interactive product catalog that draws information from one table and logs transactions in another. The s
interface options for displaying data.

This chapter considers how you can use Windows controls with data. I'll also share some advice about how to do it without h
fundamental topics:

- How to use .NET's data-binding framework with any generic control. You'll also learn about the specialized DataGrid
- How to create "data aware" controls-controls that interact efficiently and intelligently with data sources without requi
- How to use adapter classes and miscellaneous wizardry to help automate the shuffle of information through the leve

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Introducing Data Binding

Traditionally, data binding has been viewed with a great deal of suspicion. Many developers feel that it's an inflexible, clumsy

Data binding usually suffers from several well-known problems:

It's inflexible. For example, you can only bind special controls to special objects-and when you do, you usually los completely, as data controls don't allow you to participate in their work.

It's ugly. When you bind to data, you often have to display all available rows, and sacrifice any ability to format deta friendlier representation, forget it.

It's fragile. Data binding doesn't follow classic three-tier design. Instead, it binds database details directly to user in among different applications or environments, you are entirely on your own.

It's proprietary. A fine-tuned data binding solution is great-until your organization decides to upgrade to a newer pr because much of the logic is hard-coded in proprietary designer or project files.

.NET provides some remarkable data binding enhancements that just might change your mind. The first two obstacles-lack framework. The questions of application design and proprietary standards may still apply, depending on your goals. Certainly use of data binding still makes it extremely difficult to port your code to a non-Microsoft platform.

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Basic Data Binding

Almost every control in .NET supports data binding in one form or another. However, different controls support data binding i to the TextBox.Text, Button.Text, or PictureBox.Image property (although there are other possibilities, as you'll discover sho control like ListBox or CheckedListBox can hold an entire list of data or the contents of a single field from a database. Last,

You don't need to create any database code to start working with data binding. .NET allows controls to bind to any class tha

Arrays. You can use the ArrayList, Visual Basic's default Collection class, and a custom collection class that derive not supported.

ADO.NET data objects. Technically, you can only directly bind to a DataColumn, DataView, or DataViewManager o DataView it provides. Similarly, when binding to a DataSet .NET automatically uses the default DataViewManager.

Your custom classes. You have two options-you can create custom collections that implement IList or, more likely source.

Figure 9-1 shows the relationship between data providers and data consumers.

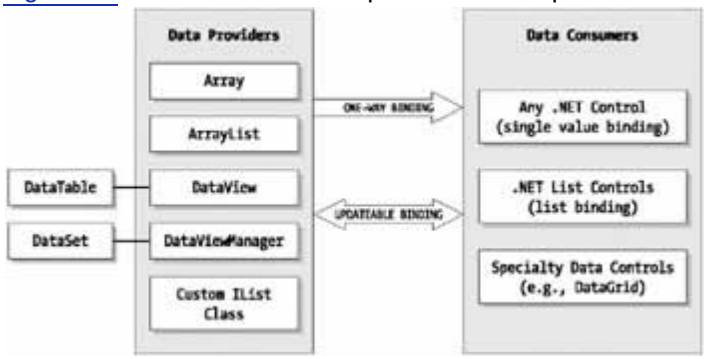


Figure 9-1: Information flow in .NET data binding

Simple List Binding

Binding to a list is one of the most common data binding tasks. All the basic .NET list controls supply a DataSource propert array, and bind it to a list using the DataSource property:

```
string[] cityChoices = {"Seattle", "New York", "Tokyo", "Montreal"};
lstCity.DataSource = cityChoices;
```

The list appears with the items from the array preloaded (see Figure 9-2).

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev

ADO.NET Data Binding

One of the most common applications of data binding is with ADO.NET data objects. This type of binding is easy, painless,

The basic principle is the same for binding collections of custom objects. However, instead of specifying a property name wi application. The result is shown in Figure 9-7.

Figure 9-7: Binding to a DataView

```
DataSet dsStore = new DataSet();
dsStore.ReadXmlSchema(Application.StartupPath + "\\store.xsd");
dsStore.ReadXml(Application.StartupPath + "\\store.xml");

lstName.DataSource = dsStore.Tables["Products"];
lstName.DisplayMember = "ModelName";
```

Note The ADO.NET examples in this chapter read DataSets from XML, as this approach doesn't clutter the examples with data access logic, and doesn't require any special relational database product. The "data aware" control examples later in this chapter separate the data logic into a discrete class, which allows this code to be completely independent of the data-binding logic.

In this example, the code appears to bind to a DataTable object, but it actually binds to the DataTable.DefaultView property. that follows doesn't change the actual information in the DataTable, but it does ensure that only a subset of it will be shown

```
DataSet dsStore = new DataSet();
dsStore.ReadXmlSchema(Application.StartupPath + "\\store.xsd");
```

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Data Binding Exposed

The secret behind data binding lies in two objects that you don't ordinarily see: BindingContext and CurrencyManager. Every zero or more CurrencyManager objects. Both objects are contained in the System.Windows.Forms namespace.

The CurrencyManager object shoulders the responsibility for tracking the user's position in the bound data and synchronizing properties, including Count, and the ever-important Position, which indicates an integer row index. It performs its work autom

The BindingContext object, on the other hand, creates CurrencyManager objects as required. Depending on how you configu data sources (or different positions in the same data source). Figure 9-14 diagrams this relationship.

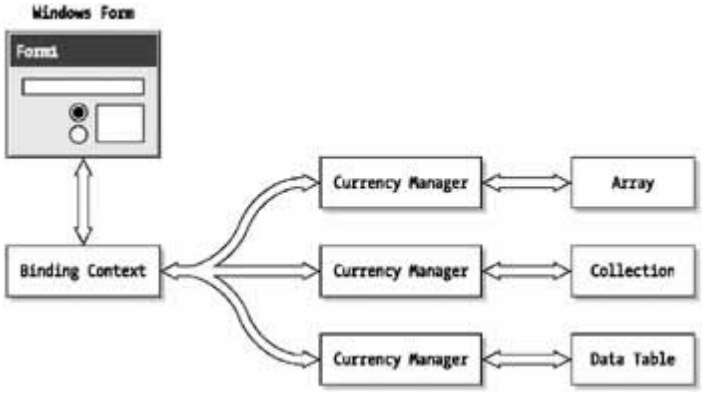


Figure 9-14: Data binding under the hood

There are really only three reasons that you might want to access the data binding objects:

- To programmatically control record navigation.
- To programmatically react to record navigation.
- To create a new BindingContext that allows you to store a different position to the same data.

Navigation with Data Binding

To navigate programmatically you need to access the form's BindingContext object, and modify its Position property. Unfort you either need to store the data source in a form-level variable, or create a form-level variable to track the binding object. Th

First, create the variable for storing the BindingContext object:

```
private BindingManagerBase storeBinding;
```

Next, in the Form.Load event handler create the bindings and store a reference to the binding object. The only new line is hig

```
private void MultipleControlBinding_Load(object sender, System.EventArgs e)
{
    DataSet dsStore = new DataSet();
    dsStore.ReadXmlSchema(Application.StartupPath + "\\store.xsd");
    dsStore.ReadXml(Application.StartupPath + "\\store.xml");

    cboModelName.DataSource = dsStore.Tables["Products"];
    cboModelName.DisplayMember = "ModelName";
```

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The DataGrid Control

The DataGrid is perfect for those who want a single data control that can do everything on its own. Even with some impressi
row). If you need a perfect super-grid control, you are likely to find that many custom options will soon appear on the networ

To use the DataGrid, you only need to assign a table or DataSet to its DataSource property. If you use an entire DataSet, th

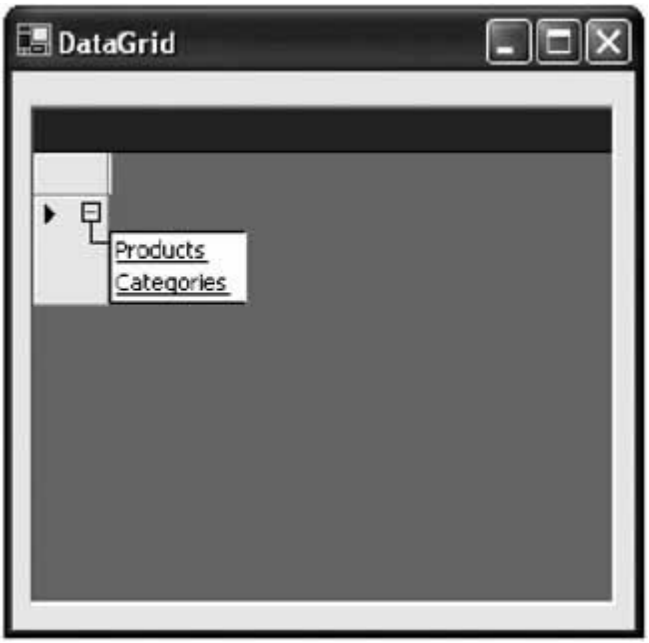


Figure 9-18: DataGrid navigation links

DataGrid Relations

You can also make use of these navigation links to create a master-detail list. All you need to do is create the appropriate ta

```
// Create a relation between categories and products.
DataRelation dr = new DataRelation("Products in this category",
  dsStore.Tables["Categories"].Columns["CategoryID"],
  dsStore.Tables["Products"].Columns["CategoryID"]);

// Add the relation to the DataSet.
dsStore.Relations.Add(dr);

// Bind the data grid.
dataGrid1.DataSource = dsStore.Tables["Categories"];
```

It's not as flexible as our custom solution for master-detail forms, but it works well with little tweaking required. Figure 9-19 s

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Encapsulation with Data Controls

With this exhaustive look at data binding, you now know how to tailor data for your user interface without writing manual cod
if you use data binding in the way I've described, you'll soon end up with a tangle of database-specific details (such as forma
typed-meaning that if a field name changes in the database, your user interface code needs immediate modifications to survi

This state of affairs is far from ideal. Luckily, there are a few ways to minimize the problem. One way is not to use data bindi
You can then fill this table with content that maps individual controls to specific data fields. A simple helper function in a data

```
public class DBHelper
{
    public static void FillForm(Form formToBind,
     DataTable mappings)
    {
        DataRow[] rowMatch();

        foreach (Control ctrl in formToBind.Controls)
        {
            // See if this menu item has a corresponding row.
            rowMatch = mappings.Select("ControlName = "' + ctrl.Text + ""');

            // If it does, configure the binding accordingly.
            if (rowMatch.GetLength(0) > 0)
            {
                // We use offset 0 because there can only be one match.
                string fieldToUse = rowMatch[0]["DataField"];

                // We assume the text property is the one to be filled.
                // Alternatively, we could add a database field with
                // this information.
                ctrl.Text = dt.Rows[fieldToUse];
            }
        }
    }
}
```

This technique works well because it establishes an extra layer of indirection between the database and the controls. It's ea
user interface is quite generic. Of course, you need to manually call this method every time the user moves to a new row to

Note Variations on this theme are examined in Chapter 14, which shows you how to use a database to roll your own context-sensitive Help.

Another way to help separate your database from your user interface code is by keeping database-specific content like field

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Data-Aware Controls

Not all controls work well with data binding. For example, the popular TreeView and ListView controls need to be filled manu control of the entire process. Maybe you want to create a control that can't be filled all at once, but uses partial data reads o

.NET provides many opportunities for data integration without data binding. One handy technique is using the Tag property. I Tag property to store any information or object you need. For example, you could use this property to store the relevant busi

The next example shows a TreeView that embeds the data it needs to use the Tag property of each node. Here's the code n

```
DataSet dsStore = new DataSet();

dsStore.ReadXmlSchema(Application.StartupPath + "\\store.xsd");
dsStore.ReadXml(Application.StartupPath + "\\store.xml");

// Define the relation.
DataRelation relCategoryProduct = new DataRelation("Products in this category",
  dsStore.Tables["Categories"].Columns["CategoryID"],
  dsStore.Tables["Products"].Columns["CategoryID"]);
dsStore.Relations.Add(relCategoryProduct);

TreeNode nodeParent, nodeChild;
foreach (DataRow rowParent in dsStore.Tables["Categories"].Rows)
{
    // Add the category node.
    nodeParent = treeDB.Nodes.Add(rowParent["CategoryName"]);

    // Store the disconnected category information.
    nodeParent.Tag = rowParent;

    foreach (DataRow rowChild in rowParent.GetChildRows(relCategoryProduct))
    {
        // Add the product order node.
        nodeChild = nodeParent.Nodes.Add(rowChild["ModelName"]);

        // Store the disconnected product information.
        nodeChild.Tag = rowChild;
    }
}
```

Chapter 9 - Data Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter has provided an in-depth examination of the inner workings of data binding. It has also considered the best pra
data source details. In Chapter 11 you revisit this issue with a twist, and examine a case where tight coupling is exactly wha

The chapter also considered some useful examples about how to sidestep data binding and create custom controls that are
excellent solution when you want to use controls that don't support data binding well, like the ListView and TreeView.

Chapter 10 - MDI Interfaces and Workspaces

byMatthew MacDonald
Apress 2002

Companion Web Site

Chapter 10: MDI Interfaces and Workspaces

Multiple document interface (MDI) represents one of the most common user interface designs on the Windows platform. MD windows like toolbars and floating palettes). MDI windows turn up in some form or another in most complex applications, an

In this chapter, you learn how to create an MDI application in .NET, and how it's dramatically easier to use and more flexible with the document-view architecture. Finally, toward the end of the chapter, you explore one of the trickiest aspects of MDI i

The Story of MDI

MDI windows have been the de facto standard for user interfaces for the past several years, although their dominance is fadi windows instead. For example, Internet Explorer is an SDI application-if you want to open several pages at once, the pages SDI application, although it was one of the earliest MDI examples in the Windows world. Most other word processors and do a unique user interface with tabbed and grouped windows, the basic principle-hosting several different windows in one large c

The debate between SDI and MDI is sometimes heated. There is no clear standard, although Microsoft officially states that S user it is designed to serve. If you are dealing with advanced users who need to manage several views simultaneously, an M you are creating a simple editor for a novice user, it's probably clearer to follow a simpler SDI design like that used in Micros

Types of MDI Applications

There are essentially two types of MDI applications:

Document applications. These applications use a single application window to contain multiple identical documen several files at once.

Workspace applications. These applications provide several different windows (which correspond to different featu program that allows you to simultaneously browse through a list of users, enter new projects, and report software bu able to perform one task at a time.

Chapter 10 - MDI Interfaces and Workspaces

byMatthew MacDonald

Apress 2002

Companion Web Site

MDI Essentials

In .NET, there is no sharp distinction between ordinary windows and MDI windows. In fact, you can transform any window int a window back and forth at will, which is a mind-expanding trick never before allowed.

```
this.IsMdiContainer = true;
```

When displayed as an MDI container, the form's surface becomes a dark gray open area where other windows can be hoste

```
Child frmChild = new Child();
frmChild.MdiParent = this;
frmChild.Show();
```

Ideally, you perform this task before you display the window, but with .NET you don't need to. In fact, you can even have mor the MdiParent property.

Figure 10-1 shows two different views of an MDI parent with a contained MDI child.

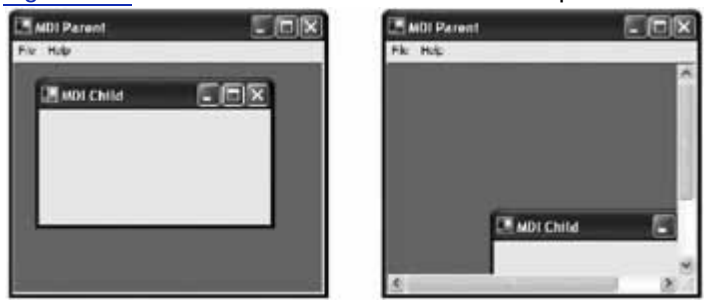


Figure 10-1: An MDI Child

One of the most unusual features of .NET MDI parents is that they can display any type of control. Traditionally, MDI parent .NET, however, you can add any other type of control, and it remains fixed in place (or anchored and docked), suspended "a

This trick can be used to create a bizarre window like that shown in Figure 10-2, or a unique type of floating tool window (alth

Chapter 10 - MDI Interfaces and Workspaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Document-View Architecture

Many developers will recognize document-view architecture as a staple of MFC design. In .NET, the emphasis is less critica
our refresh example), and don't require an additional separation between the document and the view. Tasks that typically re

On the other hand, there are several scenarios that are particularly well suited to a dedicated document-view architecture:

- When you are using complex documents.
- When you are providing more than one view of the same document.
- When you want the flexibility to provide different views in separate windows or in a single window.

When discussing MDI interfaces, a *document* is the actual underlying data. For example, with Microsoft Word the document
example, the view in Microsoft Word might just include the page that is currently being edited (which can be scrolled to differ

A typical document-view application uses the following ingredients:

- A document class.
- A document view class that references an instance of a document.
- An MDI child class that hosts the view.
- An MDI container that holds all the MDI children.

Why would a document require more than one view? It's easy to think of a view as a window onto a different part of a docum
editing view where changes are made and a print preview that shows the final layout. Both views represent the same data in
Similarly, you might have a document object that corresponds to a large amount of information from a database. You could s
example is an HTML file, which can be viewed as straight text or marked-up content.

A Document-View Ordering Program

Our next example presents a fairly sophisticated model that supports real-time previews using the document-view architectu

- An Order document object that contains a list of OrderItem objects.
- Two view objects: OrderPrintPreview and OrderGridView. Both derive from the UserControl class, but they could be
- A Child form class, which can display either of the two view objects.
- A main Parent class, which provides a toolbar and the event handling logic that creates the document objects and c
- Resource classes, like Product, which represents an individual product, and PriceList, which provides a static GetIte

Figure 10-7 shows the relationship of some of the classes in this example.

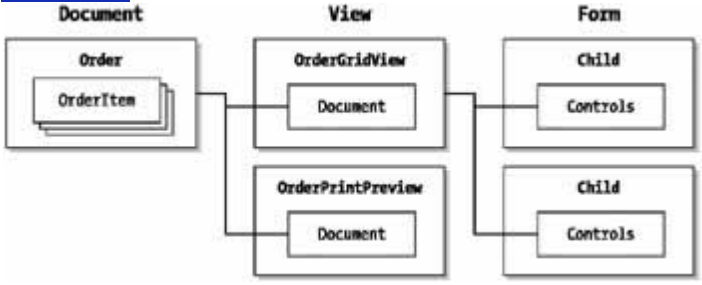


Figure 10-7: The document-view architecture in the ordering program

Document class

The heart of this application is the document class called Order, which represents a collection of items in a sales order. Bec
class, which represents an item in the catalog.

```
public class Product
```

Chapter 10 - MDI Interfaces and Workspaces

byMatthew MacDonald

Apress 2002

Companion Web Site

Floating Windows and Docking

Everyone expects an MDI interface to sport dockable windows and toolbars-floating controls that can be latched into place o awkward. Windows (and previous application frameworks) do not provide native support for most of these features. Instead, floating windows never look exactly the same in any two Windows applications-even if those two applications are both writte

Floating Toolbars

By default, when you create a toolbar it is automatically docked to the top of the form. However, it doesn't need to be-as wit

To create a toolbar that can float, you need to add some additional logic to the mouse events that detect if the toolbar is clic of a movement in either direction. Then other event-handling logic comes into play, which allows the control to be dragged ar

Here's the code that needs to be added to the Form class to support this design. It's similar to the dragging code developed process of moving the floating control.

```
private bool draggingToolbar;
private Point draggedFrom;

private void toolBar1_MouseDown(object sender,
 System.Windows.Forms.MouseEventArgs e)
{
    draggingToolbar = true;
    draggedFrom = new Point(e.X, e.Y);
    toolBar1.Capture = true;
}

private void toolBar1_MouseMove(object sender,
 System.Windows.Forms.MouseEventArgs e)
{
    if (draggingToolbar)
    {
        if (toolBar1.Dock == DockStyle.Top)
        {
            // Check if the dragging has reached the threshold.
            if (draggedFrom.X < (e.X - 20) || draggedFrom.Y < (e.Y - 20))
            {
                draggingToolbar = false;

                // Disconnect the toolbar.
```

Chapter 10 - MDI Interfaces and Workspaces

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter explored MDI programming and design. The chapter began with an introduction to .NET's effortless MDI feature more detail with a sophisticated example of document-view architecture, which provides the freedom to create multiple synch into one of MDI design's hottest topics with an introduction to coding your own dockable windows.

Chapter 11 - Dynamic User Interface

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 11: Dynamic User Interface

Overview

One of the most common questions in any Windows programming language is how to programmatically add a control to a fo
especially if the programmer needs to add a licensed ActiveX control or handle the events of the newly created control.

.NET erases the distinction between design-time and runtime control programming. In .NET programming, every control is cr
editor, Visual Studio .NET generates the appropriate code and adds it to a collapsed region of your Form class. By studying

Of course, creating a dynamic user interface is about much more than defining a control class at runtime. It's also a philosop
database records, user preferences, or localization needs. This chapter examines some innovative techniques you can use
drawing tools, document applications, and the question of localization. It even considers how you can create a custom form

Chapter 11 - Dynamic User Interface

byMatthew MacDonald

Apress 2002

Companion Web Site

The Case for Dynamic User Interface

One theme you see in this chapter is the recurring question-how dynamic should a dynamic user interface be? You'll see ex dynamically from scratch. So which approach is best?

As usual, it all depends on your project and design goals. To determine where dynamic user interface fits in, you need to co Studio .NET to create their interface include:

To design a static interface. It's far easier to create an interface that won't change for long periods of time with pe
It hides the ugly code details. These are difficult to manage due to the sheer number of lines. .NET controls do n property set statements to fully configure your controls..
It saves time. The design-time environment makes it faster to create and maintain an interface. Changes can be ap

On the other hand, there are some things that user interfaces designed in the IDE don't handle well:

If the interface must change according to certain distinct rules. In this case, you may find yourself writing a gr all its text translated to different languages. In this case, it may be a good point to consider a more radical solution more manageable.
If you need to standardize the consistency of similar but not identical visual elements (like a common dialo to creating a custom form layout engine.
If you want to create a program that allows the user to create an interface. In some cases, your product mig non-programmer to define or modify some aspects of the interface.
If you are designing a drawing or diagramming tool. You could use the GDI+ drawing features described later to a vector-based diagramming tool.

These are the types of problems that the discussion centers on in this chapter.

Chapter 11 - Dynamic User Interface

byMatthew MacDonald

Apress 2002

Companion Web Site

Creating Controls at Runtime

Creating a control at runtime involves a few simple steps:

1. Create the control object as you would any other class.
2. Set the properties for the control (including basics like size and position).
3. Add the control object to the Controls collection of a container control, like a Form, GroupBox, Panel, or TabPage.
4. If you want to handle any of the control's events, use the AddHandler statement to hook up your code.

To demonstrate this process, consider the sample button generator program shown in Figure 11-1. This program creates a b to every new button's Click event, ensuring that .NET can capture user clicks (and display a brief user message at the botto

```
public class ButtonMaker : System.Windows.Forms.Form
{
    // (Windows designer code omitted.)

    private int buttonCount = 0;

    private void cmdCreate_Click(object sender, System.EventArgs e)
    {
        buttonCount++;

        // Create the button. Button newButton = new Button();
        newButton.Text = "Button " + buttonCount.ToString();
        newButton.Left = int.Parse(txtLeft.Text);
        newButton.Top = int.Parse(txtTop.Text);

        // Attach the event handler.
        newButton.Click += new EventHandler(ButtonHandler);

        this.Controls.Add(newButton);
    }

    private void ButtonHandler(object sender, System.EventArgs e)
    {
        status.Text = " You clicked ... ";
        status.Text += ((Button)sender).Text;
    }

}
```

Chapter 11 - Dynamic User Interface

byMatthew MacDonald
Apress 2002

Companion Web Site

Dynamic Content

When discussing dynamic interfaces, it's useful to draw a distinction between those that generate controls dynamically (like
can appear in just about any situation, but it's most common in these cases:

- Applications that need to be localized or configured for different sets of users.
- Data-driven applications (like product catalogs) that use interfaces designed to closely model the organization of a d

One simple example of dynamic content is the average About box (shown in Figure 11-7). It rarely makes sense to hard-cod
guaranteed to remain correct (and it can be extremely tedious to synchronize if you use autoincrementing version numbers).

```
lblProductName.Text = Application.ProductName;
lblProductVersion.Text = "Version: " + Application.ProductVersion.ToString();
lblPath.Text = "Executing in: " + Application.ExecutablePath;
```

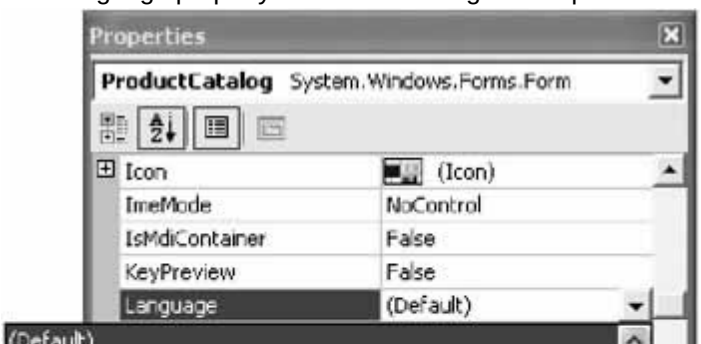
Figure 11-7: Dynamic content in the About box

Localization

The .NET platform provides considerable support for localizing Windows Forms through resource files. Using resource files,
to be translated into different languages) are embedded in a separate satellite assembly. You can create different localized v

The basic process for creating a localizable form is simple:

1. Set the Localizable property for the Form to true using the Properties window.
2. Set the Language property of the Form using the Properties window. You'll be provided with the full list of recognized



Chapter 11 - Dynamic User Interface

byMatthew MacDonald
Apress 2002

Companion Web Site

Data-Driven Programming

An extreme example of dynamic content is found in applications that generate their entire interface using the information in a data-driven applications, saving you hours of recompiling, and providing other non-programmers with the tools they need to m

Occasionally, a programmer develops a system that allows every aspect of a form to be defined in a database record or XML specialized applications might use a scaled down version of these techniques, like a survey-generation tool that allows you t is a pricing application, where the program must match the product catalog as closely as possible. This is the type of applic

A Data-Driven Pricing Application

The next example demonstrates that, contrary to what three-tier design dictates, tight coupling between the user interface a

Consider the case where you have a product catalog that changes frequently. The scope of the changes aren't limited to pric different pricing schemes and discount structures. If you write this application ordinarily, you end up creating a great deal of or later, these frequent modifications will introduce unexpected errors or make the program difficult to change or extend (for e

A more successful way to model this problem is to create a program that constructs its interface according to information in list control, with the description for each also read from the database. When the user chooses items from a category, the wh with Visual Studio 6), and text boxes are added dynamically for quantity (see Figure 11-13).

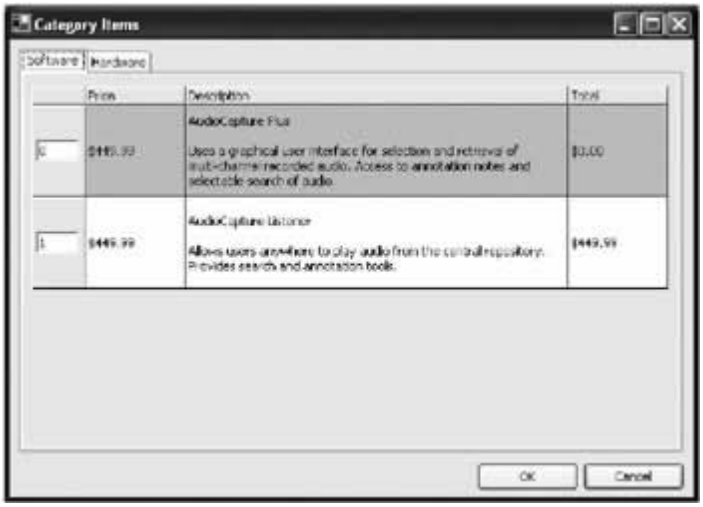


Figure 11-13: The dynamic ordering window

The pricing information is also read from a database. However, the pricing is not product-specific. Instead, every product ha schemes, and gives each one a descriptive name. The Pricing_Products table maps these two tables together, assigning ea 11-14.

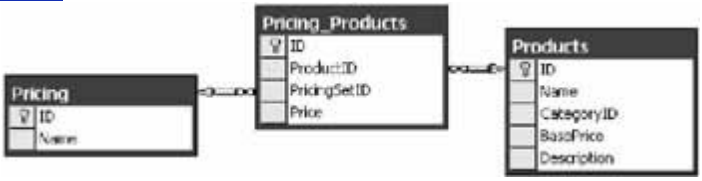


Figure 11-14: Pricing table structure

This application would typically ship with a special admin tool that is little more than a glorified database editor. It would allo system is great for salespeople who love to invent new price structures for every customer.

Note

Chapter 11 - Dynamic User Interface

byMatthew MacDonald

Apress 2002

Companion Web Site

Control Layout Engines

As you've seen, the .NET forms architecture provides support for laying out controls using coordinates in a grid. This approac environment.

However, there are times when grid layout is not necessarily the best approach. For example, you may need a container co accommodate the width of the largest control or shrinking them to fit the size of the container, to name just two possibilities you use the same layout logic in different containers (for example, forms, group boxes, and tabs). To provide a more generic

A layout manager is a class that dictates the layout of child controls in a container. Ideally, a layout manager can be reused

Note If you've programmed with Java before, the idea of layout managers is nothing new. Some of the layout managers provided for Java containers include FlowLayout (similar to a word processor), BorderLayout (which divides the screen into five zones), CardLayout (like a stack of cards layered on top of each other), GridLayout (which allows one component per equal-sized cell), and GridBagLayout (which adds support for variable control sizes and location with a grid).

Generally, a layout manager connects itself to the action by listening for layout events from the container control. It then itera layout manager, this may mean ignoring the Location property and even the Size property of each control. It could also invol

The SingleLineFlow Layout Manager

A good example of a simple layout provider is shown with the following SingleLineFlow example. It lays out one control per li display (as with a TabPage container control).

```
public class SingleLineFlow
{
    private Control container;
    private int margin;

    public SingleLineFlow(Control parent, int margin)
    {
        this.container = parent;
        this.margin = margin;

        // Attach the event handler.
        container.Layout += new LayoutEventHandler(UpdateLayout);

        // Refresh the layout.
        UpdateLayout(this, null);
    }

    public int Margin
    {
        get
```

Chapter 11 - Dynamic User Interface

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter considered dynamic user interfaces. But rather than limit the discussion to questions about how to create contr
do it), the chapter examined some more radical approaches. These techniques allow you to dynamically build a vector-based
It's remarkable what you can accomplish if you surrender the design-time comfort of the Visual Studio .NET IDE.

Many of these applications have specific niches, and the techniques discussed here aren't suitable for all applications. On th
chapter, you *need* to use a dynamic interface-without it you'll be trapped in an endless cycle of user interface tweaking and r
a chance to write innovative code-and that is always fun.

Chapter 12 - GDI+ Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 12: GDI+ Basics

Overview

.NET provides a new framework of classes for two-dimensional drawing and rendering. Taken together, these classes, found GDI+.

Technically, GDI+ still relies on the same low-level Windows APIs you may have used in Windows programming of the past. functions was that the programmer could write text and images to different devices (printers, monitors, and video cards), with and made use of any optimizations that the hardware might provide. Unfortunately, the GDI functions required a lot of coding

The GDI+ types in .NET are object-oriented wrappers over the low-level GDI functions and, strictly speaking, they don't add a support for geometric transformations, antialiasing, and palette blending. Many of these techniques required a painful amour

You've already seen GDI+ at work throughout this book. In fact, a number of the more advanced examples would have been

- In Chapter 3, you saw many of the basic GDI+ ingredients in the System.Drawing namespace, including objects rep
- In Chapter 4, you saw an owner-drawn menu that created menu entries using text and a thumbnail icon.
- In Chapter 5, you saw irregularly shaped forms that construct and outline their borders with a graphics path.
- In Chapter 6, you saw how ImageList images could be drawn on a form's surface.
- In Chapter 9, you saw a custom DataGrid column that draws text content and an icon.

This chapter explains the underlying GDI+ framework that makes all this possible.

Chapter 12 - GDI+ Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Paint Sessions with GDI+

The heart of GDI+ programming is the System.Drawing.Graphics class. The Graphics class encapsulates a GDI+ drawing su combination of the Graphics class methods.

Accessing the Graphics Object

There are essentially two ways to access a live instance of the Graphics class. In many of the examples you've looked at so object is provided as a parameter to an event handler.

For example, the code that follows draws a curve onto a form using the Graphics.DrawArc() method (see Figure 12-1):

```
// This code handles the Form.Paint event.

private void form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
    Pen drawingPen = new Pen(Color.Red, 15);
    e.Graphics.DrawArc(DrawingPen, 50, 20, 100, 200, 40, 210);
}
```

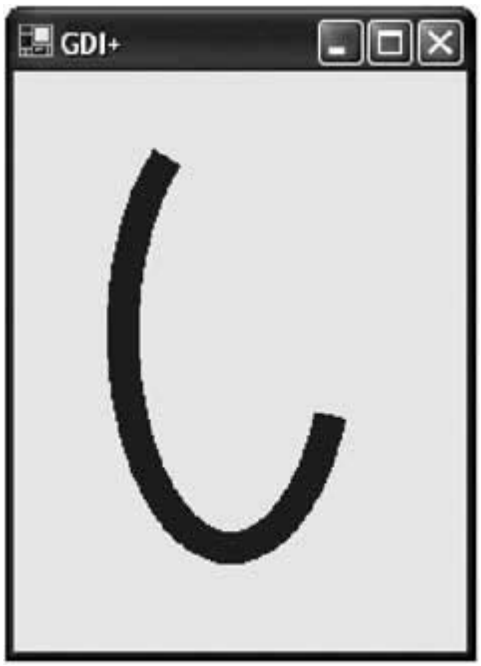


Figure 12-1: Painting to a GDI+ surface

You could perform the same task by overriding the OnPaint() method, which is the best approach for an owner-drawn control

```
// This code overrides the base Form.OnPaint() method.
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
```

Chapter 12 - GDI+ Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Optimizing GDI+ Painting

Painting is a performance-sensitive area for any application. Slow rendering may not stop your application from performing it some techniques that optimize drawing with GDI+ surfaces.

Painting and Resizing

One often overlooked fact about automatic repainting is that it only affects the portion of the window that is obscured. This is follows, which paints an ellipse that is the same size as the containing window. The result is pictured in Figure 12-3.

```
private void form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
    Pen drawingPen = new Pen(Color.Red, 15);
    e.Graphics.DrawEllipse(DrawingPen, New Rectangle(new Point(0, 0),
                           this.ClientSize));
}
```

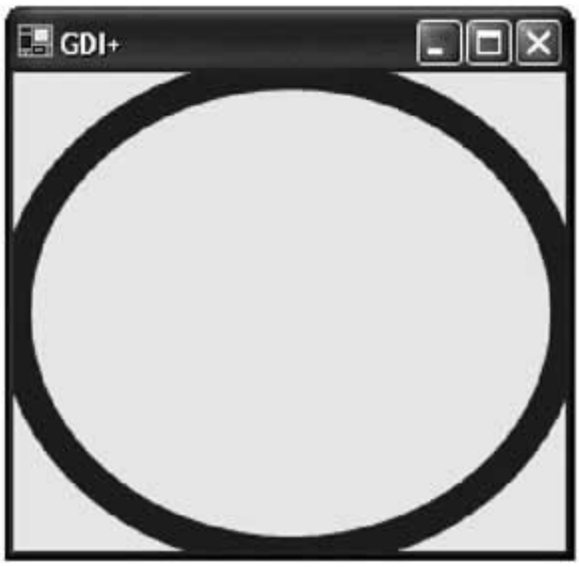
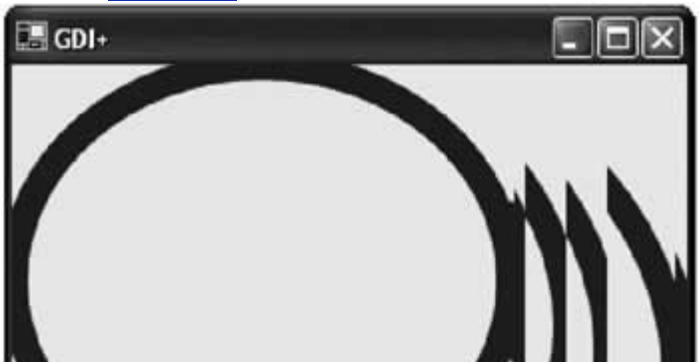


Figure 12-3: Filling a form with an ellipse

When you resize this window, you'll discover that the painting code isn't working correctly. The newly exposed portions of th shown in Figure 12-4.



Chapter 12 - GDI+ Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev

The Graphics Class

The majority of the GDI+ drawing smarts is concentrated in the Graphics class. Table 12-1 describes the basic set of Graph

Table 12-1: Basic Graphics Class Members

Member
CompositingMode and CompositingQuality
InterpolationMode
SmoothingMode and TextRenderingHint
Clear()
Dispose()
FromHdc(), FromHwnd(), and FromImage()
GetHdc() and RemoveHdc()
IsVisible()
MeasureString()
Save() and Restore()
SetClip()

The Graphics class also provides a slew of methods for drawing specific shapes, images, or text. Most of these methods be

Most of the methods in Table 12-2 are self-explanatory. Two interesting methods that I haven't described yet include DrawPa

Table 12-2: Graphics Class Methods for Drawing

Chapter 12 - GDI+ Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

Hit Testing

In Chapter 11, you saw how you could build a simple drawing application by dynamically adding controls. An alternative (and curves, and other shapes have no ability to capture mouse actions and raise the typical MouseDown and Click events. Inste determine if a shape was clicked. This process is known as hit testing.

.NET provides basic hit testing support through a Contains() method that's built into the Rectangle structure. It examines a s Rectangle.

However, there are a couple of quirks that take some getting used to with Rectangle hit testing:

A Rectangle is a combination of points (defined by a top-left corner, width, and height). It doesn't necessarily corresp the Rectangle with one of the GDI+ drawing methods.

The Rectangle is the only structure that supports hit testing. That means that if you create another shape (like a reg or retain the original Rectangle for later use.

The next example uses hit testing with the square-drawing program developed earlier. When the user right-clicks the form, t the clicked point (see Figure 12-17).

```
// Reacts to the Form.MouseDown event.
private void DrawSquare_MouseDown(object sender,
 System.Windows.Forms.MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        // Add a square and update the screen.
        Rectangle square = new Rectangle(e.X, e.Y, 20, 20);
        squares.Add(square);
        this.Invalidate(square);
    }
    else if (e.Button == MouseButtons.Right)
    {
        // Search for the clicked square.
        int squareNumber = 0;
        foreach (Rectangle square in squares)
        {
            squareNumber++;
            if (square.Contains(e.X, e.Y))
            {
               MessageBox.Show("Point inside square #" +
                               squareNumber.ToString());
            }
        }
```

Chapter 12 - GDI+ Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

The ControlPaint Class

Technically, the ControlPaint class isn't a part of GDI+. However, it's an extremely useful tool for custom control developers

This functionality is tremendously useful. For example, if you want to create a special control that contains a list of items wi to draw your own, and probably end up with a rather crude looking square. With the ControlPaint class, however, you can u you can also turn to the ControlPaint class.

The ControlPaint class consists entirely of the static methods described in Table 12-5. Here's a line of code that uses it to d

```
ControlPaint.DrawCheckBox(e.Graphics, new Rectangle(10, 10, 50, 50), ButtonState.Checked)
```

Table 12-5: Essential ControlPaint Methods

Method
DrawBorder() and DrawBorder3D()
DrawButton() and DrawCaptionButton()
DrawCheckBox()
DrawComboButton()
DrawFocusRectangle
DrawGrid()
DrawImageDisabled()
DrawLockedFrame() and DrawSelectionFrame()
DrawMenuGlyph()
DrawMixedCheckBox
DrawRadioButton()
DrawScrollButton
DrawSizeGrip()

And here's one that draws the familiar dotted focus rectangle:

Chapter 12 - GDI+ Basics

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

In this chapter you learned how to use .NET's revitalized painting framework, and the optimized techniques that make drawin
master if you want to develop your own owner-drawn controls, like hit testing and the ControlPaint class. The next chapter de

Chapter 13 - GDI+ Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 13: GDI+ Controls

Overview

Owner-drawn controls are one of the most ambitious projects a developer can undertake. This is not because they are conc
handle all aspects of its appearance. If you can create a control using composition (i.e., a user control) or by inheriting from
hand, if you need complete control over drawing and behavior, or you want to introduce some of the unusual GDI+ features t

The prime advantage to GDI+ controls is freedom. The prime disadvantage with GDI+ controls is that they aren't nearly as a
details manually:

- Scrolling support
- Focus cues (i.e., indicating when the control has focus)
- The "pushed" state appearance for a button control
- Special cues or "hot tracking" appearance changes when the mouse moves over the control
- Hit testing to determine if a click was made in an appropriate area

This chapter introduces several example controls, and shows how they confront these problems and add a few visual treats.

Chapter 13 - GDI+ Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Simple GDI+ Controls

The first type of GDI+ control that might occur to you to use is one that simply wraps one of the GDI+ drawing features you e renders a closed figure depending on the properties you set. Or, you might want to create a special type of label that paints i properties. That's the type of example considered next with the GradientLabel control.

A Gradient Label

The first example presents a special label that allows the developer to add a gradient background by choosing two colors. Th of gradient fill through an additional property. The GradientLabel control is a quick and painless way to add a label with a gra code.

The GradientLabel class inherits from UserControl and overrides the drawing logic. This isn't necessary-it could simply inher it easy to work with the GradientLabel class in a test project, rather than requiring a separate DLL and client test program.

```
public class GradientLabel : System.Windows.Forms.UserControl
{
    // (Code omitted.)
}
```

The first step is to create the required properties. In this case, you need to store information about the text to be displayed o

```
private string text;
private Color colorA = Color.LightBlue;
private Color colorB = Color.Purple;
private LinearGradientMode gradientStyle = LinearGradientMode.ForwardDiagonal;
```

Each member variable requires a separate property procedure. For brevity's sake, I've left out the optional attributes you cou property invalidates the display, ensuring that the gradient and text are repainted as needed.

```
[Browsable(true),
 DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public override string Text
{
    get
    {
        return text;
    }
    set
    {
        text = value;
        this.Invalidate();
    }
```

Chapter 13 - GDI+ Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Creating Button Controls

The label controls are fairly easy to develop because they are essentially static pieces of user interface. Other controls may box needs to handle key presses. To create one of these controls can require significant extra code and thorough testing to

To illustrate some of these considerations, the next example presents a button that's been created from scratch by deriving

A Hot Tracking Button

This example develops a HotTrackButton that displays an image and text. When the mouse is positioned over the image, it a

Figure 13-4: The HotTrackButton

This control project raises some unique, subtle challenges:

- The clickable portion of the button should only include the image. Thus, the control needs to use hit testing when a
- The button can appear in several states, including disabled, selected (when the mouse is positioned above the imag
- The button must be able to deal with any valid image size.

The first step is to create a control class that provides a member variable to track its current state. In our example, a State e

```
public class HotTrackButton : Control
{
    public enum State
    {
        Normal,
        MouseOver,
        Pushed
    }

    private State state = State.Normal;
```

Chapter 13 - GDI+ Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

Reconsidering the Vector Drawing Program

Chapter 11 developed the basic framework for a control-based drawing program using the label control. The program worked that the .NET Framework doesn't include controls for common shapes like circles, triangles, and squares. Instead, the progr

Now that you've covered GDI+, there's a far better solution available, and it only takes a little bit more coding. The basic con place of the bordered label. The drawing framework handles the dragging, resizing, and coloring automatically by setting pro

There is a possible problem. If the user draws a circle, you want the circle shape to act like a circle for all mouse operations to move the control. Similarly, this "invisible" portion of the control shouldn't overwrite other controls on the drawing surface.

Figure 13-5 shows a drawing program with shapes that doesn't take this into account.

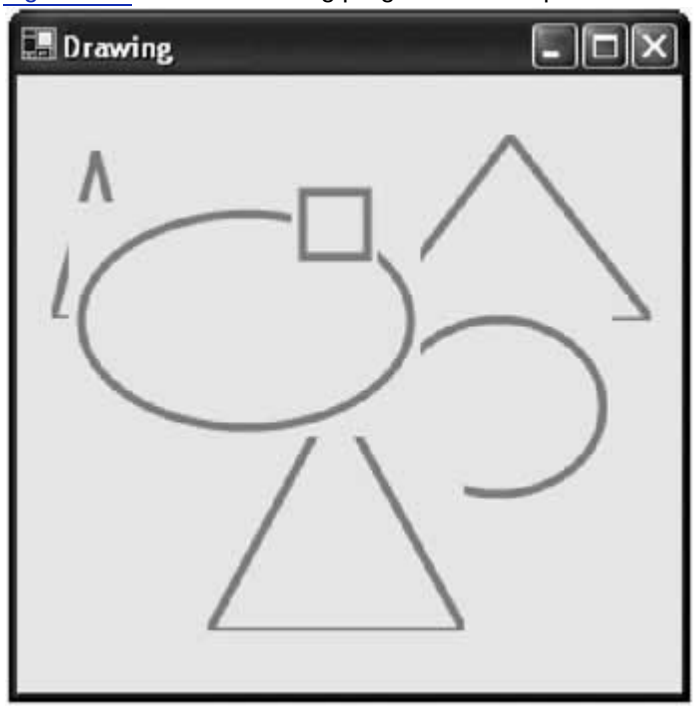
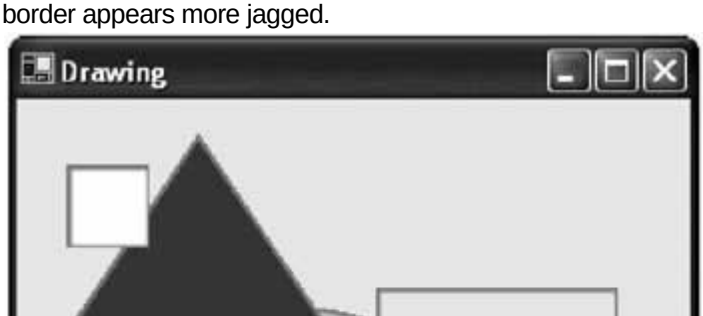


Figure 13-5: A flawed drawing program

Solving the Bounding Problem

Luckily, .NET makes it easy to create a control that has a nonrectangular bounding area. In fact, you saw this technique in C new clipping region.

Figure 13-6 shows a drawing program that uses the region property to define control borders. Note that this does have a side border appears more jagged.



Chapter 13 - GDI+ Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

A Simple Graphing Control

The last control considered here is a simple bar chart. It's a good demonstration of how you can create a higher-level GDI+ c supplied data.

The basis of the chart is a BarItem class that stores information for a single bar. This information consists of a numerical val

```
public class BarItem
{
    public string ShortForm;
    public decimal Value;

    public BarItem(string shortForm, decimal value)
    {
        this.ShortForm = shortForm;
        this.Value = value;
    }
}
```

The data for a bar chart is made up of a collection of BarItem objects. Thus, to support the control you can create a strongly collection with the control, but you would not be able to prevent the user from adding invalid objects to the chart data.

```
public class BarItemCollection : CollectionBase
{
    public void Add(BarItem item)
    {
        this.List.Add(item);
    }

    public void Remove(int index)
    {
        // Check to see if there is an item at the supplied index.
        if ((index > Count - 1) || (index < 0 ))
        {
            throw new System.IndexOutOfRangeException();
        }
        else
        {
             this.List.RemoveAt(index);
        }
```

Chapter 13 - GDI+ Controls

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

GDI+ controls represent the fusion of two remarkable new features: the nextgeneration drawing framework GDI+, and .NET's limitless, and major tool vendors have already begun developing .NET equivalents for common UI widgets like advanced scro cost.

If you want to master control development, the best approach is to review as many code examples as possible. There's no l require as much sheer code as a basic sales ordering database application. Microsoft, on its MSDN home page, provides se

The next chapter shifts the focus to a new topic: integrating Help into your .NET applications, and creating intelligent interfac

Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

Chapter 14: Help and Application-Embedded Support

Overview

Help: Is it the final polish on a professional application or a time-consuming chore? It all depends on the audience, but most confused or disoriented. Without this basic aid, you (or your organization's technical support department) are sure to be buri

In this chapter, you learn:

How to integrate Windows Help files into your applications. You look at launching Help manually, and using the cont
When to design your own Help, and how you can weave it into an application using an extensible database-based or
How to break down the limits of Help and design application-embedded support: user assistance that's integrated in
some basic approaches like affordances and wizards (through visual inheritance), and a few advanced techniques lik

Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

The Case for Help

In recent years there has been a shift away from printed documentation. The occasional weighty manual (like the book you're business software no longer assumes the user is willing to perform any additional reading. Instead, these programs are heav occasional user question.

Online Help doesn't have to take the form of a second-rate user manual, however. The advantages of online Help are remarka

Increased control. With a little effort, you can determine exactly what information users see when they click the F1 a third-party "…For Dummies" guide, and you have no way of knowing what they will find.

Rich media. With online Help you can use as many pictures as you want, in any combination, and even include so

Search tools and context-sensitivity. Help systems can automate most of the drudgery associated with finding in index), and programs can use context-sensitivity to make sure users see the appropriate topic immediately.

All help standards provide these advantages in one form or another. In the next section, you explore the Help landscape.

Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

Types of Help

Standardized Help has existed since the Windows platform was created, and even in the DOS world in little known tools like through several distinct stages, which are described in the next few sections.

WinHelp

The WinHelp format used in Windows 3.0 still exists, and can be used in the most modern Windows application. Unfortunate

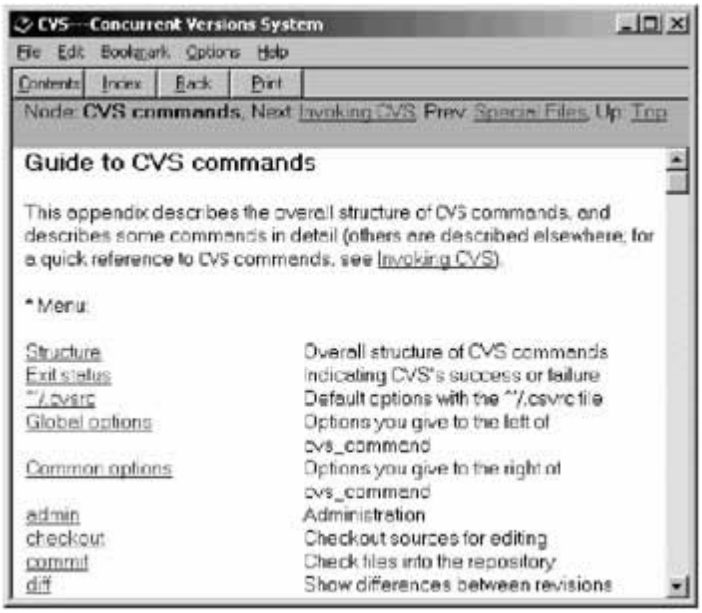
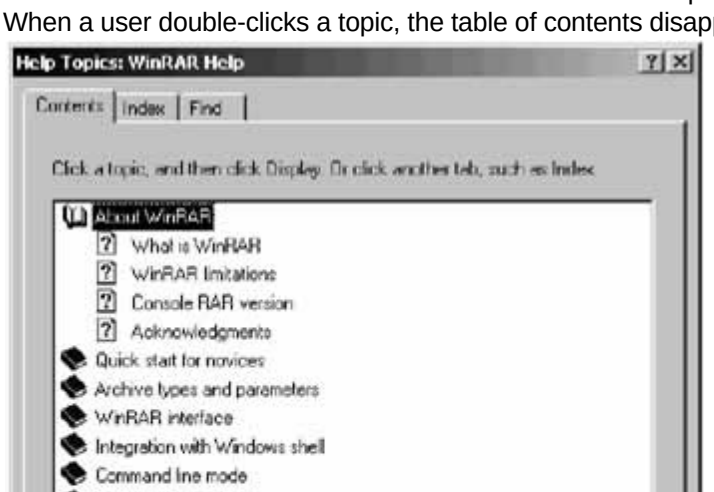

CVS---Concurrent Versions System
File Edit Bookmark Options Help
Contents Index Back Print
Node: CVS commands, Next: Invoking CVS, Prev: Special Files, Up: Top
Guide to CVS commands
This appendix describes the overall structure of CVS commands, and describes some commands in detail (others are described elsewhere; for a quick reference to CVS commands, see Invoking CVS).
* Menu:
Structure Overall structure of CVS commands
Exit status Indicating CVS's success or failure
~/.cvsrc Default options with the ~/.cvsrc file
Global options Options you give to the left of cvs_command
Common options Options you give to the right of cvs_command
admin Administration
checkout Checkout sources for editing
commit Check files into the repository
diff Show differences between revisions

Figure 14-1: WinHelp: a piece of living history

WinHelp 95

When Windows 95 was introduced, a new standard (often referred to as WinHelp 95) took over. WinHelp 95 files are familia

WinHelp 95 was a major improvement in the Help world. Whereas the original WinHelp forced developers to create their ow define the standardized multilevel table of contents. WinHelp 95 really has two parts: the .cnt contents page (which also pro When a user double-clicks a topic, the table of contents disappears, and is replaced with the appropriate help window.



Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

Classic Bad Help

Have you ever had this experience? You find an unusual option buried deep in an application, and it piques your curiosity. Y the description provided by the context-sensitive Help system: "To enable option X, click once on the option X check box. To changes."

Clearly something is missing here. You want to know what option X does; the Help wants to explain, in oddly explicit detail, the way to use a check box is an instinctive part of every computer user's understanding. If you don't know how to use a che

This is a classic example of bad help. Some of the characteristics of bad Help include:

It describes the user interface. Users don't need to know how the interface works-they will often discover that by t
It's long. Even HTML Help doesn't have the same bandwidth as a printed document, and endless scrolling is sure to
It uses visual clues. Instructing the user to look at the "top left" or "middle right" may seem logical enough, but with
It omits information. Printed documents can afford to choose what they cover. However, Help documents are shipp or setting that's in the interface.

To understand good help, you need to recognize that most Help is designed to provide reference information. Help really sh of information about a specific window or setting. This is the type of information that all users need occasionally while workin

On the other hand, Help is relatively poor at providing tutorial-based learning, which explains tasks that incorporate many dif task-based learning is generally frustrating for a number of reasons-users can't see the help window at the same time they lo are needed, and most users don't want to read a large amount of information from the computer screen anyway.

When creating Help, you should aim to divide it into discrete topics that describe individual windows, complete with all their

Tip

Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

Basic Help with the HelpProvider

One easy way to use Help with a .NET application is by adding the HelpProvider control. This "invisible" control shows up in

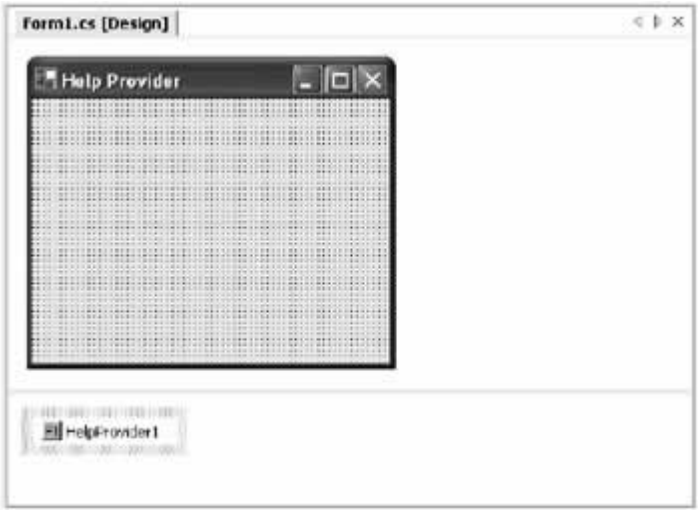


Figure 14-5: The HelpProvider

The HelpProvider uses a basic HelpNamespace property that sets the help source. This could be a path to an ordinary .hlp f the designer (with a name like ShowHelp on HelpProvider). Set this to true, and the help file is automatically launched if the

You can also connect or disconnect help manually in code, using the HelpProvider.SetShowHelp() method. Just pass the co

```
private void HelpTest_Load(object sender, System.EventArgs e)
{
    // This turns on F1 help support for cmdAdd.
    hlp.SetShowHelp(cmdAdd, true);

    // This disables it.
    hlp.SetShowHelp(cmdAdd, false);
}
```

Note There really isn't any difference between using the SetShowHelp() method and the extended ShowHelp property provided by the designer.With providers, Visual Studio .NET simply translates your selections into the appropriate method calls, and adds the code to the form code. So when you set the ShowHelp property, you are still in fact using the SetShowHelp() method.

You can also set the HelpKeyword and HelpNavigator properties in conjunction to configure which topic is shown when the H

Table 14-1: Values for the HelpNavigator Enumeration

HelpNavigator

Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

Invoking Help Programmatically

The examples so far require the user to press the F1 key. This automated approach doesn't work as well if you want to provi reminder can reassure the user that help is nearby.

To trigger help programmatically, you need to use the static ShowHelp() method of the Help class (found in the System.Win uses the Help class behind the scenes when the user presses F1.

There are several overloaded versions of the ShowHelp() method. The simplest requires a help filename (or URL) and the co operating system reasons). Here's an example that shows the test.hlp file:

```
Help.ShowHelp(this, "test.hlp");
```

Additionally, you can use a version of the ShowHelp() method that requires a HelpNavigator, one that requires a keyword, or context-sensitive help:

```
Help.ShowHelp(this, "test.hlp", HelpNavigator.Topic, "about.htm");
```

To save yourself some work when using this technique with the HelpProvider, you would probably retrieve these values from

```
private void cmdHelp_Click(object sender, System.EventArgs e)
{
    Help.ShowHelp(this, hlp.HelpNamespace, hlp.GetHelpNavigator(this),
                  hlp.GetHelpKeyword(this));
}
```

Similarly, you might use a right-click context menu for a control that provides the control's default help:

```
private void mnuHelp_Click(object sender, System.EventArgs e)
{
    Control ctrl = mnuLabel.SourceControl;
    Help.ShowHelp(ctrl, hlp.HelpNamespace, hlp.GetHelpNavigator(ctrl),
                  hlp.GetHelpKeyword(ctrl));
}
```

This menu event handler is written using the SourceControl property, which means it's generic enough to be used with any Help keyword.

Help Without the HelpProvider

Now that you are this far, it's possible to unshackle yourself completely from the HelpProvider class. It works like this-handle launch the appropriate help programmatically with the Help class.

There are two tricks to making this work. The first one is setting the form's KeyPreview property to true, which makes sure i make sure you create an event handler for the KeyDown event, not the KeyPress event, which doesn't react to the special F

Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

Application-Embedded Support

One of the crucial shortcomings with the Help systems you have looked at is that they are all designed to provide fine-graine miserably when it tries to walk the user through a long, involved task. However, better help is possible. What's needed is a c

Application-embedded support represents that change. Quite simply, embedded help is user assistance that is a first class r complete. Embedded help provides far greater user assistance, but also requires far more development work.

Some examples of embedded help include:

Process-oriented. Some applications reserve a portion of their interface for continuous tip messages, or use a tiny trains users as they work, and is used to great effect in fairly complex computer games (like the popular hit The Sim

Stationary embedded. This is the most common form of embedded help, and it refers to the content added to dial

Agents. This is one of the most advanced and time-consuming types of embedded help. It was pioneered largely by schizophrenic-it provides tools aimed to make it easy for all developers to use this level of support, but it only occas

Bidirectional help. To some, this is a holy grail of embedded help, but it's rarely achieved, and usually only in a pr in two ways. Not only does the user have to leave the application to read most Help files, but once the appropriate in file to act on the user's behalf, and actually show the user how to do what is needed. Some Help files do provide ruc communication is difficult and fragile. With bidirectional help, Help can perform the necessary task, once it determin

Affordances

Affordances are the visual "clues" designed to demystify a complex application. For example, Windows uses brief descriptio

Figure 14-10: A dialog with affordances

Help and affordances represent a bit of a paradox. Nothing can clarify a confusing dialog box more than a couple of well-plac anything else that requires reading. They either try to figure out the task on their own or, in the case of an error, repeat the ta

Chapter 14 - Help and Application-Embedded Support

byMatthew MacDonald

Apress 2002

Companion Web Site

The Last Word

This chapter began with the built-in .NET Framework support for Help, and then began to expand on it with integrated help a demonstrating that COM is far from gone, even after the .NET revolution.

Help strategies and systems vary widely depending on the intended audience and the application design. In this chapter we t used in every scenario. Instead, it helps to keep some basic principles in mind:

A help file should describe the purpose of various settings, not how to use common controls. No one needs an exp box.

The best affordances are descriptive labels, not instructions. No dialog box has the space or formatting power of a p

The best error is one that doesn't happen. It may take more effort in your code to disable or hide invalid options, but

Help must be context-sensitive. A confused user won't search through a Help file to find a relevant topic-a printed do

Perform usability tests. When writing a program, you design based on who you *believe* the audience is. At some po

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

Symbols

& (ampersand) to configure shortcut keys, 71

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

A

Abort() method of DataGridColumnStyle class, 383
About box, 449-450
AcceptButton for form, 146
AcceptsReturn property of TextBox control, 95
AcceptsTab property of TextBox control, 95
access keys for controls, 71
Action property of TreeViewEventArgs object, 227
Activate() method of form, 403
Activated event of Form class, 146
Activation property of ListView control, 211
Active property of ToolTipProvider control, 86
ActiveControl property of ContainerControl class, 154
ActiveLinkColor property of LinkLabel control, 93
ActiveMdiChild property of MDI container, 402
ActiveX controls, 259
imported, 89-90
MSFlexGrid, 458
Add() method of Nodes collection, 225
Add Reference dialog box, 262
AddArc() method of GraphicsPath object, 484
AddBezier() method of GraphicsPath object, 484
AddBeziers() method of GraphicsPath object, 484
AddClosedCurve() method of GraphicsPath object, 484
AddCurve() method of GraphicsPath object, 484
AddEllipse() method of GraphicsPath object, 484
AddLine() method of GraphicsPath object, 484
AddLines() method of GraphicsPath object, 484
AddOwnedForm() method of Form class, 162
AddPath() method of GraphicsPath object, 484
AddPie() method of GraphicsPath object, 484
AddPolygon() method of GraphicsPath object, 484
AddRange() method
of Controls class, 39
of Nodes collection, 225
AddRectangle() method of GraphicsPath object, 484
AddRectangles() method of GraphicsPath object, 484
AddString() method of GraphicsPath object, 484
AddToDock() method of MDIMain object, 433
administrative utilities, 12-13
ADO.NET data binding, 360-373
advanced conversions, 370-372
formatting data before, 367-369
multiple control binding, 363-365
row validation and changes, 372-373
updating, 365-367
ADO.NET data objects, as data source, 351
affordances, 551-553
AfterCheck event of TreeView Node, 226

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

B

BackColor property of controls, 66
background color
for GDI+ drawing, 481
gradient for, 502-505
of square in drawing, 448
background painting, disabling, 477
background process, status bar information about, 249
BackgroundImage property of Control object, 80
bar chart, control to create, 525-530
BarBreak property of MenuItem class, 115
BarChanged event, 529
BarItem class, 525-526
base class, choosing, 298-299
BeforeCheck event of TreeView Node, 226
BeforeCollapse event of TreeView Node, 226
BeforeExpand event of TreeView Node, 226
BeforeLabelEdit event of ListView control, 216
BeforeSelect event of TreeView Node, 226
BeginEdit() method of node, 215
BeginUpdate() method of ListView control, 219
Bezier curve, drawing, 551
binding. *See* data binding
Binding object, events, 366
BindingContext object, 373-374
creating additional, 379-380
bitmap format
drawing program, 443
Image.FromFile() method to read, 80
for TextureBrush, 493-494
for Toolbox, 316-317
bitmap thumbnail viewer, 276-289
enhancements and threading, 286-289
events, 284-286
private variables, 278
refresh options, 282
testing, 283-284
BoldedDates property of MonthCalendar control, 109
border
hatched for Panel control, 431-432
for selected PictureBox, 285
BorderLayout manager, 460
BorderStyle property
of StatusBarPanel object, 251
of ToolBar control, 243
Bounds property of Screen class, 150
Break property of MenuItem class, 115
breakpoints, 322
Browsable control property attribute, 228

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

C

Calendar control in ASP.NET, 331
CalendarFont property of DateTimePicker control, 107
CalendarForeColor property of DateTimePicker control, 107
CalendarMonthBackground property of DateTimePicker control, 107
CalendarTitleBackColor property of DateTimePicker control, 107
CalendarTitleForeColor property of DateTimePicker control, 107
CalendarTrailingForeColor property of DateTimePicker control, 107
CancelButton for form, 146
CanExtend() method of IExtenderProvider interface, 299
CanUndo property of TextBox control, 95
Capture property of Control object, 427
CardLayout manager, 460
cascading windows in MDI applications, 405-406
Category control property attribute, 312
CenterParent StartPosition value, 148
CenterScreen StartPosition value, 148
Changed event, 75
CharacterCasing property of TextBox control, 96
charting ActiveX control, 89
check boxes, drawing, 498
CheckBox controls, 96-97
CheckBoxes property
of ListView control, 212
of TreeView control, 228
Checked property
of DateTimePicker control, 108
of MenuItem class, 115
of TreeNode collection, 229
CheckedIndices property of ListView control, 212
CheckedItems property of ListView control, 212
CheckedListBox control, 97
CheckOnClick property for list controls, 98
CheckState property of CheckBox control, 96
child form class in MDI application, 421-422
.chm file, 535
Class Library project, 261-263
classes, 24-28
controls as, 29-30
.NET for user interface, 29-39
roles of, 24-25
and types, 25-28
Clear() method
of Graphics object, 481
of Nodes collection, 225
of TextBox control, 96
Click event
for BitmapViewer, 284-286

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

D

DashPattern property of pen, 489
DashStyle property of pen, 490
data-aware controls, 391-398
data binding
ADO.NET, 360-373
advanced conversions, 370-372
formatting data before, 367-369
multiple control binding, 363-365
row validation and changes, 372-373
updating, 365-367
in ASP.NET, 20
basics, 350-359
binding context creation, 379-380
BindingContext object and CurrencyManager object, 373-374
master-detail forms, 377-378
navigation, 374-375
reacting to record navigation, 376-377
DataGrid control, 380-387
column mapping, 382-383
custom column styles, 383-387
relations, 381
encapsulation, 387-390
interfaces for, 365
lists to complex objects, 353-356
and ListView control, 397-398
problems, 350
simple list binding, 352-353
single-value binding, 357-360
validation of bound data, 388-390
data browser, 14
data-driven programming, 457-459
data-driven user interfaces, 46
data layer in three-tier design, 49
problems, 49
data transfer plan, defining, 45
data types, Properties window support for, 335
database
help based on, 545-546
integer comparison with enumeration, 370
separation from user interface, 387-388
storing records linked to graphics, 371
table to map user levels for permissions, 453
TreeView object for browser, 51-52
DataBindings collection, of controls, 357-360
DataGrid control, 380-387
anchoring, 175
column mapping, 382-383

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

E

Edit() method of DataGridColumnStyle class, 383
editing, overriding methods to prevent, 383-384
Editor attribute, 338
EditValue() method of UITypeEditor class, 340
ellipse
drawing, 523-524
filling, 483
email address, validating with regular expression, 140-141
enabled controls, 409
Enabled property
of Control object, 74
of LinkLabel.Link object, 94
of MenuItem class, 115
of ToolBarButton object, 245
encapsulation, 44-46
custom control decisions, 275
data binding, 387-390
EndCap property of pen, 489
EndUpdate() method of ListView control, 219-220
EnsureVisible() method
of ListView control, 213
of TreeNode collection, 231
enumerations, 44
AnchorStyles, 174
comparison of database integers with, 370
custom, 335-336
FormStartPosition, 148-149
HelpNavigator, 540
ShapeType, 520
State, for buttons, 513
Enum.Parse() method, 67-68
error checking. *See also* 336
error messages, user response to, 17-18
ErrorProvider control, 298
validation with, 138-140
event handler
for controls, 40-41
overriding in inherited form, 197-198
events
for custom controls, filtering, 327-329
Form class, 146-147
exception classes, 276
Expand() method of TreeNode collection, 230
ExpandAll() method of TreeNode collection, 230
expanding TreeView levels, 230-231
extender providers
control layout engines as, 463

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

F

FAR, 537
feedback for user, 19
ProgressBar control for, 104
FileNameEditor, 337
filenames, converting for storage in database record, 371
fill
brushes for, 490-493
for line, 489
FillClosedCurve() method of Graphics object, 483
FillEllipse() method of Graphics object, 483
FillPath() method of Graphics object, 521
FillPie() method of Graphics object, 483
FillPolygon() method of Graphics object, 483
FillRectangle() method of Graphics object, 483
FillRectangles() method of Graphics object, 483
FillRegion() method of Graphics object, 483
filtering properties and events for custom controls, 327-329
FindForm() method of Control object, 71
FirstDayOfWeek property of MonthCalendar control, 110
flicker, 473
reducing, 477
floating toolbars, 425-428
FlowLayout manager, 460
focus, 72-74
Focus() method of Control object, 74
focus rectangle, drawing, 498
Focused property of Control object, 74
FocusedItem property of ListView control, 211
Font property
of Control object, 69
of StatusBar control, 250
FontDialog, 170
FontEditor, 337
FontHeight property of Control class, 70
fonts
antialiasing with, 476-477
converting string to object, 358
determining installed, 70
for drawing text, 469
for owner-drawn menus, 124
recommendation for applications, 69
substitution by .NET platform, 70
FontViewer utility, 70
ForeColor property of controls, 66
ForeFront Help Center, 537
form-based help, 542-543
limitations, 546
Form class, 143-164

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

G

GAC (Global Assembly Cache), 265-268
installing control to, 267-268
GDI (Graphics Device Interface), 465
GDI+
brushes, 490-493
ControlPaint class, 498-500
coordinate systems and transformations, 485-487
Graphics class, 481-493
drawing methods, 482-483
GraphicsPath methods, 484
members, 481-482
hit testing, 494-497
nonrectangular shapes, 496-497
for owner-drawn controls, 188
paint sessions, 466-469
painting and debugging, 480
painting optimization, 470-480
double buffering, 477-480
rendering mode and antialiasing, 475-477
and resizing, 470-471
window portions, 472-475
pens, 488-490
GDI+ controls
button controls, 512-518
details to handle, 501
graphing control, 525-530
simple, 501-512
gradient label, 502-505
gradient label design-time support, 505-508
marquee label, 509-512
vector-based drawing program, 519-525
GetBounds() method of Screen class, 150
GetChildAtPoint() method of Control object, 72
GetContextMenu() method of Menu class, 113
GetEditStyle() method of UITypeEditor class, 340
GetForm() method of ContextMenu class, 114
GetHdc() method of Graphics object, 481
GetImages() method for bitmap thumbnail viewer, 287-288
GetItemAt() method of ListView control, 213
GetLicense() method for custom LicenseProvider, 344-346
GetMainMenu() method of Menu class, 113
GetMinimumHeight() method of DataGridColumnStyle class, 383
GetMinimumSize() method of DataGridColumnStyle class, 383
GetNextControl() method of Control object, 74
GetNodeAt() method of TreeView control, 231
GetNodeCount() method of TreeView control, 222
GetPaintValueSupported() method of UITypeEditor class, 340
GetPreferredHeight() method of DataGridColumnStyle class, 383

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

H

Handle property
of Control class, 81
of Menu class, 113
Handles keyword, and derived event handlers, 198
HasChildren property of Control object, 71
Hashtable class, 300
HatchBrush class, 491-492
hatched border for Panel control, 431-432
HeaderStyle property of ListView control, 212
help
ActiveX controls for, 89
application-embedded support, 550-559
affordances, 551-553
agents, 553-559
user interface integration, 553
authoring tools, 536-537
case for, 531-532
characteristics of bad, 537-538
control-based and form-based, 542-543
creating, 548-550
database-based, 545-546
invoking programmatically, 543-550
task-based, 546-548
text on StatusBar, 252-253
types, 532-537
HTML Help, 534-535
MS Help 535-536
WinHelp, 532
WinHelp 533-534
HelpButton property of Form class, 145
helper classes, 25
HelpIconProvider control, 303-307
HelpIconProvider.vb file, 321
HelpNamespace property, 541
HelpNavigator enumeration, 540
HelpProvider control, 539-543
help without, 544-545
HelpTest project, 543
hiding Panel control, 431
history of user interface design, 3-10
command-line, 3-5
GUI, 8-10
menu-driven model, 7-8
question-answer model, 5-7
hit testing, 494-497
nonrectangular shapes, 496-497
HKEY_CURRENT_USER, 151
HKEY_LOCAL_MACHINE, 151

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

I

IBindingList interface, 365
IComparer class for ListView column sort, 213-214
IComponentChangeService interface, 334
methods, 335
Icon property of Form class, 145
of NotifyIcon, 257
of StatusBarPanel object, 251
icons
drawing, 482
for invisible controls, 84
resource files for, 318
for Toolbox, 316-317
for TreeView nodes, 229-230
IDataErrorInfo interface, 365
IEditableObject interface, 365
IExtenderProvider interface, 298
IList interface, 365
Image object
converting file name to, 371
for double buffering, 478
FromFile() method, 80
GetThumbnail() method, 286
properties and methods for controls, 80-81
Image property
of button, 514
of Control object, 80
of Label control, 91
of PictureBox control, 97
ImageAlign property
of Control object, 80
of Label control, 91
ImageEditor, 337
ImageIndex property
of Control object, 80
of TabPage, 256
of ToolBarButton object, 245
ImageList collection, 201-204
code for, 203-205
method to fill, 204
properties, 202
ImageList property
of controls, 204
of TabControl, 256
of ToolBar control, 243
ImageListStreamer class, 203
images
storing records linked to, 371
transparent regions for, 202

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

J

jagged edges, antialiasing to smooth, 475-477
Java, layout managers for containers, 460
JavaScript, 535

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

K

keyboard, control response to, 75-79
KeyDown event of Control object, 75
KeyPress event of Control object, 75
KeyPreview property, 544
KeyUp event of Control object, 75

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

L

Label control, 90-91
with gradient background, 502-505
ProgressBar control with, 271-276
TabIndex property, 73-74
LabelEdit property of ListView control, 215-216
LabelWrap property of ListView control, 212
Language property of form, 452
LargeIcon for ListView View enumeration, 210
LargeImageList property of ListView control, 211
layout manager for controls, 459-463
LayoutMdi() method, 405-406
learning, transferrable, 9
Length property of LinkLabel.Link object, 94
LIC file, 341-342
LicenseKey property, overriding for custom license, 344
LicenseProvider attribute, 342
licensing
for ActiveX controls, 89
custom controls, 341-346
LicFileLicenseProvider class, 341-342
LinearGradientBrush class, 492-493
LineJoin property of pen, 489
lines, drawing, 482
LinkArea property of LinkLabel control, 93
LinkBehavior property of LinkLabel control, 93
LinkColor property of LinkLabel control, 93
LinkData property of LinkLabel.Link object, 94
LinkLabel control, 91-94
Links property of LinkLabel control, 93
LinkVisited property of LinkLabel control, 93
List control, 97-101
data binding, 352-353
to complex objects, 353-356
DataBindings collection, 358
filling with known colors, 68-69
SelectedIndex event, 468
List for ListView View enumeration, 210
ListBox control, 97
anchoring, 175
IntegralHeight property, 174
ListView control, 206-217
adding information, 216
basic members, 211
and data binding, 397-398
vs. DataGrid object, 48
label editing, 215-216
reordering columns, 215

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

M

main menus, 111
attaching, 118
MainMenu class, 114
for synchronizing with ToolBar, 247-249
manifest file for Windows XP styles of forms, 163-164
Manual StartPosition value, 148
marquee label, 509-512
Mask property of controls, 297
masked TextBox control, 294-297
master-detail forms, 377-378
DataGrid control for, 381-382
MaxDate property
of DateTimePicker control, 108
of MonthCalendar control, 110
MaxDropDownItems property for ComboBox control, 100
MaximizeBox property of Form class, 144
maximized MDI child forms, 402
Maximum property of NumericUpDown control, 104
maximum size of form, 172-173
MaxLength property
of ComboBox control, 100
of TextBox control, 96
MaxSelectionCount property of MonthCalendar control, 110
MDI (Multiple Document Interface) applications
dockable windows, 429-434
with owner-drawn controls, 429
with timers, 430-434
document-view architecture, 410-424
child form class, 421-422
document class, 412-416
OrderGridView class, 416-418
OrderPrintPreview class, 418-420
parent form class, 422-424
essentials, 400-410
interface state, 408-410
layout, 405-406
merging menus, 407
relatives, 402
synchronizing children, 403-404
floating toolbars, 425-428
and resizable windows, 172
story of, 399-400
types, 400
MdiChildren property of form, 403
MDIMainController, 409
MdiParent property of form, 403
Me keyword, 35
MeasureItem event of MenuItem class, 122

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

N

Name property of controls, 62
NamedImage class for bitmap thumbnail viewer, 280
namespace, importing, 262
navigation
BindingContext object for, 374-375
in DataGrid control, 381
reacting to record navigation, 376-377
.NET object libraries, importance of learning, 60
.NET platform
consistency in, 11
custom controls, 259
data binding, 349-350
drawing optimization, 480
font substitution, 70
menu objects in, 111
predefined colors, 66
support for dialog forms, 157
types, 25-28
user interface classes, 29-39
vs. Windows API, 58-59
New Project dialog box, for Class Library creation, 261
Node property of TreeViewEventArgs object, 227
Nodes collection
manipulating, 223-226
selecting nodes, 226-228
in TreeView control, 218-219
Nodes property of TreeView control, 328
NotifyIcon, 257
NotifyParentProperty control property attribute, 313
novice users, 14
NumericUpDown control, 103

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

O

object-oriented programming, encapsulation, 44-46
objects, 24-28
choosing for extending, 299
in list control Items property, 100
Offset() method for Rectangle object, 495
OLE color code, 67
OnClick() method, overriding, 518
OnDocumentChanged() method, 414
OnKeyPress() method, 39
online examples. *See* project examples
online help, advantages, 532
OnMouseDown() method, overriding for button, 515-516
OnMouseLeave() method, overriding for button, 515
OnMouseMove() method, overriding for button, 515
OnMouseUp() method, overriding for button, 515-516
OnPaint() method, 469
for MarqueeLabel control, 510-511
OnPaintBackground() method
for MarqueeLabel control, 510-511
overriding for gradient fill, 504
overriding to reduce flicker, 477
OnSelect() method, overriding, 254
Opacity property of Form class, 145
Open method in MDI interface, 415
OpenFileDialog, 170
ordering program, 411-424
child form class, 421-422
document class, 412-416
OrderGridView class, 416-418
OrderPrintPreview class, 418-420
parent form class, 422-424
organizational controls, 104
OwnedForms property of Form class, 162
owner-drawn controls, 260
benefits, 518
dockable windows with, 429
GDI+ for, 188
overriding OnPaint() method for, 467
owner-drawn menus, 122-130
Owner property of Form class, 162
OwnerDraw property of MenuItem class, 122
ownership of forms, 161-163

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

P

Padding property of TabControl, 256
PageSetupDialog, 170
PageUnit property of Graphics object, 485
Paint event handler, 469
Paint event of Control object, 80
Paint() method of DataGridColumnStyle class, 383
painting
and debugging, 480
optimization, 470-480
double buffering, 477-480
rendering mode and antialiasing, 475-477
and resizing, 470-471
window portions, 472-475
sessions, 466-469
and refreshing, 467-469
painting programs, 443
PaintValue() method
overriding for gradient fill thumbnail image, 507-508
of UITypeEditor class, 339
Panel control, 104
property to hide or show, 431
PanelClick event for StatusBar, 252
panels, docking with, 181-183
parent form class in MDI application, 422-424
Parent property of Control object, 71
ParentForm property of ContainerControl class, 154
ParenthesizePropertyName control property attribute, 313
PartialPush property of ToolBarButton object, 245
PasswordChar property of TextBox control, 96
Paste() method of TextBox control, 96
pens, 488-490
PenType property of pen, 489
performance
custom controls and, 276
for painting, 470
PerformClick() method
of Button control, 94
of MenuItem class, 116
PerformSelect() method of MenuItem class, 116
permissions
database table to map levels, 453
hiding or disabling controls based on, 457
PictureBox controls, 97
pictures. *See* images
pie shape
drawing, 483
filling, 483

Index

byMatthew MacDonald

Apress 2002

Companion Web Site

Index

Q

question-answer user interface, 5-7

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

R

radio button
container for, 104
drawing, 499
RadioButton controls, 96-97
RadioCheck property of MenuItem class, 115
read-only DataGrid, overriding methods to prevent editing, 383-384
ReadOnly control property attribute, 313
ReadOnly property of TextBox control, 96
RebuildChart() method, 527-528
recompiling projects, 195
records in database, process for adding, 365
RecreatingHandle property of Control object, 82
rectangle
drawing, 483
filling, 483
hit testing, 494-495
references, adding to project, 262
referencing custom controls, 263-265
Refresh() method of Control object, 80
refresh, with double buffering, 478
RefreshChildren() method, 403-404
RefreshList() method for PrintPreview control, 418
RefreshProperties control property attribute, 313
RefreshText() method of MDI child window, 404
RegEx object, 141
RegexTypeEditor, 338
Region object, filling, 483
Region property
of controls, 520-525
of form, 186
registration of ActiveX controls, 89
RegistryForm class, 151
regular expressions
RegexTypeEditor for, 338
validation with, 140-142
relative-based navigation of TreeView control, 222
Remove() method
for controls, 32
of Nodes collection, 225
RemoveAll() method of ToolTipProvider control, 86
RemovedOwnedForm() method of Form class, 162
RemoveHdc() method of Graphics object, 481
rendering mode, 475-477
rendering quality for GDI+ drawing, 481
RenderTransparent property of Label control, 91
repainting window, 467-469
when debugging, 480
representation of document, view as, 410

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

S

Save method in MDI interface, 415
Save() method of Graphics object, 482
SaveFileDialog, 169
SaveSize() method of RegistryForm class, 151
Screen class, 149
screen resolution
and form size, 171-172
and window location decisions, 149
scroll button, drawing, 499
scrollable forms, 152-154
thumbnail viewer for bitmaps as, 276-283
ScrollableControl class
dock padding, 178
Form class inheritance from, 152-153
ScrollChange property of MonthCalendar control, 110
ScrollPixelAmount property for MarqueeLabel control, 509
ScrollTimerInterval property for MarqueeLabel control, 509
SDI (single-document interface), 399
segmenting information, 16
Select event of MenuItem class, 117
Select() method of TextBox control, 96
SelectAll() method of TextBox control, 96
SelectedIndex event for list controls, 468
SelectedIndex property
of DomainUpDown control, 102
for list controls, 365
of TabControl, 256
SelectedIndexChanged event of TabControl, 256
SelectedIndices collection for list controls, 99
SelectedIndices property of ListView control, 211
SelectedItem property of ListView control, 211
SelectedItemIndexChanged event of ListView control, 211
SelectedItems collection for list controls, 99
SelectedObject property of PropertyGrid control, 325-326
SelectedTab property of TabControl, 256
SelectedText property of TextBox control, 96
selection frame, drawing, 499
SelectionEnd property of MonthCalendar control, 110
SelectionLength property of TextBox control, 96
SelectionMode property for list controls, 99
SelectionRange property of MonthCalendar control, 110
SelectionStart property
of MonthCalendar control, 110
of TextBox control, 96
SelectNextControl() method of Control object, 74
serialization of control properties, 314-316
SetClip() method of Graphics object, 482
SetError() method of ErrorProvider control, 128

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

T

tab key, focus change sequence, 72-74
TabControl, 254-256
TabCount property of TabControl, 256
TabIndex property of Control object, 73
table of contents, in WinHelp 533
Table property of DataView class, 362
tables in DataSet object, 52
TabPage control, properties, 256
TabPages property of Control object, 254
TabStop property of Control object, 73
Tag property
of Control object, 62
of controls, 391-392
of ToolBarButton object, 245
Tahoma font, 69
task-based help, 546-548
testing custom controls, 321-326
for design mode, 323-325
text boxes
connecting to array, 357
customized, 33-34
setting limits, 18
Text property
of button, 514
of Control object, 69-70
of DateTimePicker control, 108
of DomainUpDown control, 102
of GDI+ control, 504
of list controls, 365
of ListItem, 213
of MarqueeLabel control, 509
of MenuItem class, 116
of NotifyIcon, 257
of StatusBar control, 250
of StatusBarPanel object, 251
of TabPage, 256
of ToolBarButton object, 245
TextAlign property of ToolBar control, 243
TextBox controls, 94-96
masked, 294-297
TextRenderingHint property of Graphics object, 481
TextureBrush class, 493-494
this keyword, 195
threads
for asynchronous printing, 421
in BitmapViewer, 286-289
modal windows and, 155

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

U

UITypeEditor classes, 507-508
custom, 339-341
prebuilt, 337-338
UML (Unified Modeling Language) diagram, 275
for BitmapViewer, 283
Undo() method of TextBox control, 96
UnitCost variable, formatting data before binding, 367-369
UNIX command-line interface, 3
UpdateDisplay() method for bitmap thumbnail viewer, 281
UpdateLayout() method for layout manager, 461
usability tests, by Microsoft, 11
UseMnemonic property of Label control, 91
user controls, 269-276
bitmap thumbnail viewer, 276-289
creating, 269-270
inheritance, 269-270
vs. inherited controls, 290
progress indicators, 271-276
size of, 270
user input, validation, 135-142
user interface design. *See also* dynamic user interface
complexity, 15-17
creativity vs. convention, 10-15
data-driven, 46
database separation from, 387-388
help integration, 553
history, 3-10
command-line, 3-5
GUI, 8-10
menu-driven model, 7-8
question-answer model, 5-7
importance, 518
inductive, 17
qualities, 3
responsibilities of, 2-3
restrictions on user, 17-20
tasks, 1
for the Web, 20
user permissions
database table to map levels, 453
hiding or disabling controls based on, 457
UserControl class, 272
GradientLabel class inheritance from, 502
users
knowledge of, 14-15
response to agents, 553
restrictions on, 17-20

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

V

Validate() method of LicenseManager class, 342
Validated event, 135-136
Validating event, 135-136
validation, 135-142
of bound data, 388-390
and business objects, 53
in database updates, 372-373
for enumerations, 336
with ErrorProvider control, 138-140
example, 136-137
with regular expressions, 140-142
Value property
of DateTimePicker control, 108
of ProgressBar control, 104
of TrackBar control, 104
variables, public class, vs. properties, 276
vector-based drawing program, 519-525
Verbs property, 329
versioning, interfaces for, 28
view in MDI interface, 410
updating, 414
View-Mediator design pattern, 40-41
View property of ListView control, 211
view styles for ListView, 210
switching between, 208
Visible property
of Control object, 74
of MenuItem class, 116
of NotifyIcon, 257
of ToolBarButton object, 245
Visited property of LinkLabel.Link object, 94
VisitedLinkColor property of LinkLabel control, 93
visual inheritance, 193-198
and ancestor form, 195
overriding event handler, 197-198
Visual Studio .NET
adding ActiveX control to project, 86-90
code generation, 37-39
Command window, 4
control layout, 66
for interface creation, 435-436
Menu designer, 118
MSDN help in, 535
opening two instances for testing, 321-322
rebuilding client project, 263
tab order settings for controls, 73

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

W

Warp() method of GraphicsPath object, 484
Web Browser control, 89
Web controls, 29
Web Forms, controls, 32
Web pages, programming user interface for, 20
WebWorks Publisher, 537
What's This help, 541
Widen() method of GraphicsPath object, 484
Width property of StatusBarPanel object, 251
WIMP, 8
Win32 color code, 67
windows
associating control with, 31
dockable, 429-434
multiple, 434
with owner-drawn controls, 429
with timers, 430-434
splitting, 180-186
collapsible, 184-185
for multiple views of same data, 183
tiling and cascading in MDI application, 405-406
Windows (Microsoft)
graphics in, 467
methods for drawing interface elements, 498
wizards, 5
Windows 534
Windows API, 58
access to, 81-82
in .NET applications, 59
vs. .NET platform, 58-59
Windows Control Project, 261
Windows controls, 29
Windows Forms, forms
problems of earlier alternative, 57-58
resource files for localizing, 450
Windows XP
forms' styles, 163-164
MonthCalendar control and, 110
WindowsDefaultBound StartPosition value, 148
WindowsDefaultLocation StartPosition value, 148
WindowState property of Form class, 145
WinHelp, 532
WinHelp 533-534
Winres.exe utility, 452
wizards, 551
form inheritance to create, 193-194
WndProc() method of Control object, 82
WorkingArea property of Screen class, 150

Index

byMatthew MacDonald
Apress 2002

Companion Web Site

Index

X

XML
for .resx file, 318
DataSet object and, 52

List of Figures

byMatthew MacDonald
Apress 2002

Companion Web Site

List of Figures

Chapter 1: Creating Usable Interfaces

Figure 1-1: The archetypal command-line interface

Figure 1-2: The command line in Visual Studio .NET

Figure 1-3: A genuinely useful wizard

Figure 1-4: A metaphor-based calendar and organizer

Figure 1-5: Explorer-like user interface

Figure 1-6: Two approaches to an audio recorder

Chapter 2: Designing with Classes and Tiers

Figure 2-1: A sample form

Figure 2-2: Control containment for a sample form

Figure 2-3: Two ways to interact with controls

Figure 2-4: Ordinary forms and visual inheritance

Figure 2-5: Using form and application switchboards

Figure 2-6: Three-tier design

Figure 2-7: An intelligent TreeView database browser

Chapter 3: Control Class Basics

Figure 3-1: Control hierarchy

Figure 3-2: A medley of different controls

Figure 3-3: Control measurements

Figure 3-4: A color changing form

Figure 3-5: A simple font viewer

Figure 3-6: Focused buttons

Figure 3-7: The Visual Studio .NET tab order tool

Figure 3-8: An event tracker

Figure 3-9: Common control picture support

List of Tables

byMatthew MacDonald

Apress 2002

Companion Web Site

Prev

List of Tables

Chapter 3: Control Class Basics

Table 3-1: Basic Control Members

Table 3-2: Members for Control Relationships

Table 3-3: Members for Dealing with Focus at Runtime

Table 3-4: Events for Reacting to the Keyboard

Table 3-5: Events for Reacting to the Mouse

Table 3-6: Graphics-Related Members

Table 3-7: Low-Level Members

Chapter 4: Classic Controls

Table 4-1: ToolTipProvider Members

Table 4-2: Label Properties

Table 4-3: LinkLabel Properties

Table 4-4: LinkLabel.Link Properties

Table 4-5: Special Button Members

Table 4-6: TextBox Members

Table 4-7: List Control Properties

Table 4-8: Special ComboBox Properties

Table 4-9: DateTimePicker Properties

Table 4-10: MonthCalendar Properties

Table 4-11: Members of the Abstract Menu Class

Table 4-12: ContextMenu Members

Table 4-13: MainMenu Members

Table 4-14: MenuItem Members

Table 4-15: .NET Validation

Table 4-16: Regular Expression Metacharacters

Chapter 5: Forms

User Interfaces in C#: Windows Forms and Custom Controls

byMatthew MacDonald ISBN:1590590457

Apress 2002 (586 pages)

Including a comprehensive examination of the user interface controls and classes in .NET, this resource provides an overview of how to design elegant user interfaces the average user can understand.

Companion Web Site

Table of Contents Back Cover

Back Cover

Describes the architecture of a well-planned user interface-everything from data binding strategies to multiple document view architecture using MDI

Teaches GDI+ essentials such as hit testing, irregularly shaped forms, owner-drawn menus, and animated controls

Shows how to integrate "intelligence" into your user interfaces with embedded help and user agents

User Interfaces in C#: Windows Forms and Custom Controls goes beyond simply covering the Windows Forms namespaces by combining a careful treatment of the API with a detailed discussion of good user-interface design principles. The combination will show you how to create the next generation of software applications using the .NET Framework. After reading *User Interfaces in C#: Windows Forms and Custom Controls*, you'll know how to design state-of-the-art application interfaces, as well as how to extend .NET controls, create data-binding strategies, program graphics, and much more.

This book contains the following:

An overview of how to design elegant user interfaces the average user can understand

A comprehensive examination of the user interface controls and classes in .NET

Best practices and design tips for coding user interfaces and integrating help

Although this book isn't a reference, it does contain detailed discussions about every user interface element you'll use on a regular basis. But you won't just learn how to use .NET controls-you'll learn how and why to extend them, with owner-drawn menus, irregularly shaped forms, and custom controls tailored for specific types of data. As a developer, you need to know more than how to add a control to a window. You also need to know how to create an entire user interface framework that's scalable, flexible, and reusable.

About the Author

Matthew MacDonald is an author, education and MCSD developer. He's also a regular contributor to programming journals, such as *Inside Visual Basic* and *C# Today*, and he's the author of several books about .NET programming, including *User Interfaces in VB .NET: Windows Forms and Custom Controls* (Apress, 2002), *The Book of VB .NET* and *ASP.NET: The Complete Reference.*